

# ADMM LP decoding of non-binary LDPC codes in $\mathbb{F}_{2^m}^*$

Xishuo Liu <sup>†</sup>, Stark C. Draper <sup>‡</sup>

July 29, 2015

## Abstract

In this paper, we develop efficient decoders for non-binary low-density parity-check (LDPC) codes using the alternating direction method of multipliers (ADMM). We apply ADMM to two decoding problems. The first problem is linear programming (LP) decoding. In order to develop an efficient algorithm, we focus on non-binary codes in fields of characteristic two. This allows us to transform each constraint in  $\mathbb{F}_{2^m}$  to a set of constraints in  $\mathbb{F}_2$  that has a factor graph representation. Applying ADMM to the LP decoding problem results in two types of non-trivial sub-routines. The first type requires us to solve an unconstrained quadratic program. We solve this problem efficiently by leveraging new results obtained from studying the above factor graphs. The second type requires Euclidean projection onto polytopes that are studied in the literature, a projection that can be solved efficiently using off-the-shelf techniques, which scale linearly in the dimension of the vector to project. ADMM LP decoding scales linearly with block length, linearly with check degree, and quadratically with field size. The second problem we consider is a penalized LP decoding problem. This problem is obtained by incorporating a penalty term into the LP decoding objective. The purpose of the penalty term is to make non-integer solutions (pseudocodewords) more expensive and hence to improve decoding performance. The ADMM algorithm for the penalized LP problem requires Euclidean projection onto a polytope formed by embedding the constraints specified by the non-binary single parity-check code, which can be solved by applying the ADMM technique to the resulting quadratic program. Empirically, this decoder achieves a much reduced error rate than LP decoding at low signal-to-noise ratios.

## 1 Introduction

Using optimization theory to solve channel decoding problems dates back at least to [1], where Breitbach *et al.* form an integer programming problem and solve it using a branch-and-bound approach. Linear programming (LP) decoding of binary low-density parity-check (LDPC) codes is introduced by Feldman *et al.* in [2]. Feldman's LP decoding problem is defined by a linear objective function and a set of linear constraints. The linear objective function is constructed using the log-likelihood ratios and is the same objective as in maximum likelihood (ML) decoding. The set of

---

\*This material was presented in part at the IEEE 2014 Int. Symp. on Inf. Theory (ISIT), Honolulu, HI, July, 2014. This work was supported by the National Science Foundation (NSF) under Grants CCF-1217058 and by a Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Research Grant. This paper was submitted to *IEEE Trans. Inf. Theory*.

<sup>†</sup>X. Liu is with the Dept. of Electrical and Computer Engineering, University of Wisconsin, Madison, WI 53706 (e-mail: xishuo.liu@wisc.edu).

<sup>‡</sup>S. C. Draper is with the Dept. of Electrical and Computer Engineering, University of Toronto, ON M5S 3G4, Canada (e-mail: stark.draper@utoronto.ca).

constraints is obtained by first relaxing each parity-check constraint to a convex polytope called the “parity polytope” and then intersecting all constraints. Compared to classic belief propagation (BP) decoding, LP decoding is far more amenable to analysis. Many decoding guarantees (e.g., [3–7]) have been developed for LP decoding in the literature. However, LP decoding suffers from high computational complexity when approached using generic solvers. Since [2], many works have sought to reduce the computational complexity of LP decoding of binary LDPC codes [8–14]. Of particular relevance is [12] where the authors use the alternating direction method of multipliers (ADMM) to decompose the LP decoding problem into simpler sub-problems and develop an efficient decoder that has complexity comparable to the sum-product BP decoder. In a more recent work [15], Liu *et al.* use ADMM to try to solve a penalized LP decoding objective and term the problem *ADMM penalized decoding*.<sup>1</sup> The penalized LP objective is constructed by adding a non-convex term to the LP decoding objective to penalize pseudocodewords, to which LP decoding can fail. Empirically, ADMM penalized decoding outperforms ADMM LP decoding in terms of both word-error-rate (WER) and computational complexity. There are two important lessons from [12] and [15]. First, ADMM based decoding algorithms can be efficient algorithms that are promising for practical implementation. Second, these algorithms can achieve lower WERs than BP does at both low and high signal-to-noise ratios (SNRs). In particular, simulation results in [15] show that ADMM penalized decoding can outperform BP decoding at low SNRs and does not suffer from an error-floor at WERs above  $10^{-10}$  for the [2640, 1320] “Margulis” LDPC code.

LP decoding of non-binary LDPC codes is introduced by Flanagan *et al.* in [16]. Results in [12] and [15] motivate us to extend ADMM decoding to non-binary codes. However, this extension is non-trivial. One key component of the ADMM decoder in [12] is a sub-routine that projects a length- $d$  vector onto the parity polytope – the convex hull of all length- $d$  binary vectors of even parity. The projection algorithm developed in [12] has a complexity of  $O(d \log d)$ . More recently, [13] and [14] propose more efficient projection algorithms, e.g., the algorithm proposed in [14] has linear complexity in dimension  $d$ . Unfortunately, these techniques cannot be directly applied to LP decoding of non-binary codes as the polytope induced has two major differences. First, unlike the binary case where 0, 1 can be directly embedded into the real space, elements of  $\mathbb{F}_q$  need to be mapped to binary vectors of dimension at least  $q - 1$  [16]. Second, permutations of a codeword of non-binary single parity-check (SPC) codes are not necessarily codewords whereas for binary SPC codes all permutations of a codeword are codewords.<sup>2</sup>

We briefly describe our approach to developing an efficient LP decoder using ADMM. We focus on non-binary codes in fields of characteristic two and represent each element in  $\mathbb{F}_{2^m}$  by a binary vector of dimension  $m$ , where each entry corresponds to a coefficient of the polynomial representing that element. This allows us to build a connection between an element’s embedding (i.e., those defined in [16]) and its binary vector representation. We then obtain a factor graph characterization of valid embeddings. With this characterization, the polytopes considered in [16] can be further relaxed to intersections of simplexes and parity polytopes. We develop two sets of important results pertinent to this relaxation. First, we show several properties of the aforementioned factor graph of embeddings. As we will see in the main text, these properties are crucial for developing an efficient ADMM update algorithm. Further, these properties are useful in determining the computational complexity of the decoder. Second, we regain the permutation properties by introducing a rotation

---

<sup>1</sup>We write “try to” because there is no guarantee that ADMM solves the non-convex program, in contrast to LP decoding, for which ADMM is guaranteed to produce the global optimum of the LP.

<sup>2</sup>Permutation symmetry is key to the algorithms in [12] and [15].

step similar to the one used in the study of BP decoding of non-binary codes in [17]. This rotation step is generic and can be applied to any finite field. It rotates the polytope so that it has a “block permutation” property. This result is not only an interesting theoretical characterization of the polytope but also plays an important role in the algorithm.

Several other works also address the complexity of LP decoding of non-binary LDPC codes. For example, in [18] and [19], algorithms are presented that perform coordinate-ascent on an approximation of the dual of the original LP decoding problem. Using binary vectors to represent elements of  $\mathbb{F}_{2^m}$  for LP decoding is considered in [20] and [21]. In particular in [20], Honda and Yamamoto introduced a subtle connection between a constant weight embedding of fields of characteristic two and the binary vector representation of fields of characteristic two in [20]. Although the authors of [20] did not provide an efficient algorithm to leverage this connection, we think that this work is important and inspiring, and build off it herein.

We list below our contributions in this paper.

- We extend the ideas of [20] and establish connections between Flanagan’s embedding of finite fields [16] and the binary vector representation of finite fields. We introduce a new factor graph representation of non-binary constraints based on embeddings (Section 3.2).
- We study the geometry of the LP decoding constraints. We show that the polytope formed from embeddings of the non-binary single parity-check code can be rotated so that the vertices have a “block permutation” property. In addition, we compare two relaxations in terms of their tightness (Section 4). A conjecture that characterizes the polytope for  $\mathbb{F}_{2^2}$  is discussed in Appendix 12.
- In the context of LP decoding, we conduct an extensive study of both Flanagan’s embedding method and the constant-weight embedding from [20]. The main result is that LP decoding using either embedding method achieves the same WER performance (Section 5.2).
- We formulate two ADMM decoding algorithms. The formulation in Section 5.3 applies to LP decoding and to penalized decoding (cf. Section 6) of non-binary codes in arbitrary fields. However, in arbitrary fields we have not been successful in developing a computationally simple projection sub-routine. In Section 5.4 we limit our focus to LP decoding of codes in  $\mathbb{F}_{2^m}$  and develop an efficient algorithm.
- We show through numerical results that the penalized decoder improves the WER performance significantly at low SNRs (Section 7).

## 2 Notation and definitions

In this section, we fix some notation and definitions that will be used through out the paper.

### 2.1 General font conventions

We use  $\mathbb{R}$  to denote the set of real numbers and  $\mathbb{F}_q$  to denote finite fields of size  $q$ . We denote by  $\mathbb{F}_{2^m}$  fields of characteristic two. We use bold capital letters to denote matrices and bold small letters to denote vectors. We use non-bold small letters to denote entries of matrices and vectors. For example,  $\mathbf{X}$  is a matrix and  $x_{ij}$  is the entry at the  $i$ -th row and the  $j$ -th column.  $\mathbf{x}$  is a vector and its  $i$ -th entry is  $x_i$ . We use script capital letters to denote discrete sets and use  $|\cdot|$

to denote the cardinality of a set. Further, we denote by  $[n]$  the set  $\{1 \dots, n\}$ . We use sans-serif font to represent mappings and functions, e.g.  $\mathbf{f}(\cdot)$  and  $\mathbf{F}(\cdot)$ . We also explicitly define the following operations:  $\text{conv}(\cdot)$  is the operation of taking the convex hull of a set in  $\mathbb{R}^n$ . It is defined as

$$\text{conv}(\mathcal{S}) := \left\{ \sum_{i=1}^{|\mathcal{S}|} \alpha_i \mathbf{s}_i \mid \sum_{i=1}^{|\mathcal{S}|} \alpha_i = 1 \text{ and } \alpha_i \geq 0 \text{ for all } i \right\}.$$

The operation  $\text{vec}(\cdot)$  vectorizes a matrix, i.e., it stacks all columns of a matrix to form a column vector. For example, let  $\mathbf{X}$  be a  $m \times n$  matrix; then  $\text{vec}(\mathbf{X}) = (x_{11}, x_{21}, \dots, x_{1m}, x_{21}, x_{22} \dots, x_{m(n-1)}, x_{1n}, \dots, x_{mn})^T$ .

## 2.2 Notation and definitions of convex geometries

Throughout this paper we use roman blackboard bold symbols to denote convex sets, e.g.,  $\mathbb{P}$ .<sup>3</sup> In particular, we define the following polytopes that will be used extensively throughout this paper:

**Definition 1** Let  $\mathbb{P}_d$  denote the parity polytope of dimension  $d$ ,

$$\mathbb{P}_d := \text{conv}(\{\mathbf{e} \in \{0, 1\}^d \mid \|\mathbf{e}\|_1 \text{ is even}\}). \quad (2.1)$$

Let  $\mathbb{S}_{d-1}$  denote the positive orthant of a unit  $\ell_1$  ball of dimension  $d-1$ ,

$$\mathbb{S}_{d-1} := \left\{ (x_1, \dots, x_{d-1}) \in \mathbb{R}^{d-1} \mid \sum_{i=1}^{d-1} x_i \leq 1, \text{ and } x_i \geq 0 \text{ for all } i \right\}. \quad (2.2)$$

Let  $\mathbb{S}'_d$  denote the standard  $d$ -simplex,

$$\mathbb{S}'_d := \left\{ (x_0, \dots, x_{d-1}) \in \mathbb{R}^d \mid \sum_{i=0}^{d-1} x_i = 1, \text{ and } x_i \geq 0 \text{ for all } i \right\}. \quad (2.3)$$

It is easy to verify that  $(x_1, \dots, x_{d-1}) \in \mathbb{S}_{d-1}$  implies  $(x_0, \dots, x_{d-1}) \in \mathbb{S}'_d$  where  $x_0 = 1 - (\sum_{i=1}^{d-1} x_i)$ . Because of this relationship, we use  $\mathbb{S}_{d-1}$  with one dimension less than  $\mathbb{S}'_d$ . Further, we let the vector index start from 0 for vectors in  $\mathbb{S}'_d$  and 1 for vectors in  $\mathbb{S}_{d-1}$ .

## 2.3 Notation of non-binary linear codes

We consider non-binary linear codes of length  $N$  with  $M$  checks, and use  $\mathbf{H}$  to denote the parity-check matrix of a code where each entry  $h_{ji} \in \mathbb{F}_q$ . We use  $j$  to represent a check node index and  $i$  to represent a variable node index. Then the set of codewords is  $\{\mathbf{c} \mid \mathbf{H}\mathbf{c} = 0 \text{ in } \mathbb{F}_q\}$ . When represented by a factor graph, the set of variable and check nodes are denoted by  $\mathcal{I}$  and  $\mathcal{J}$  respectively. Let  $\mathcal{N}_v(i)$  and  $\mathcal{N}_c(j)$  be the respective set of neighboring nodes of variable node  $i$  and of check node  $j$ .

## 2.4 Integer representation of finite fields

In this paper we often consider finite fields of characteristic two. Since we will have more occasion to add in finite fields than to multiply, we represent each element of  $\mathbb{F}_{2^m}$  using an integer in  $\{0, \dots, 2^m - 1\}$ . Without loss of generality, an element  $\alpha \in \mathbb{F}_{2^m}$  can be represented by a polynomial  $p(x) = \sum_{i=1}^m b_i x^{i-1}$ . In this paper we often use the integer representation of  $\alpha$  which is given by  $p(2) = \sum_{i=1}^m b_i 2^{i-1}$ .

---

<sup>3</sup>Note that  $\mathbb{R}$  (for real numbers) and  $\mathbb{F}$  (for finite fields) are two special cases where we use roman blackboard bold symbols.

	polynomial	bits ( $b_3b_2b_1$ )	integer
$\xi^0$	1	001	1
$\xi^1$	$\xi$	010	2
$\xi^2$	$\xi^2$	100	4
$\xi^3$	$\xi + 1$	011	3
$\xi^4$	$\xi^2 + \xi$	110	6
$\xi^5$	$\xi^2 + \xi + 1$	111	7
$\xi^6$	$\xi^2 + 1$	101	5

**Table 1:** Different representations of elements in  $\mathbb{F}_{2^3}$ .

	polynomial	bits ( $b_2b_1$ )	integer
$\xi^0$	1	01	1
$\xi^1$	$\xi$	10	2
$\xi^2$	$\xi + 1$	11	3

**Table 2:** Different representations of elements in  $\mathbb{F}_{2^2}$ .

**Example 2** We consider  $\mathbb{F}_{2^3}$  and let the primitive polynomial be  $x^3 + x + 1$ . Let  $\xi$  be the primitive element of the field. We list different representations of elements in  $\mathbb{F}_{2^3}$  in Table 1. Note that although we use the integer representations, multiplications are still performed using field operations. For example, using Table 1,  $4 \cdot 6 = 5$  in  $\mathbb{F}_{2^3}$ .

In  $\mathbb{F}_{2^2}$  we let the primitive polynomial be  $x^2 + x + 1$ . Let  $\xi$  be the primitive element of the field. We then obtain different representations of elements in  $\mathbb{F}_{2^2}$  in Table 2.

### 3 Embedding methods and representations of non-binary single parity-check codes

Embedding finite fields into reals is a crucial step in the LP decoding of non-binary codes. In this section, we first review two somewhat similar embedding methods. We refer these embeddings as “Flanagan’s embedding” and “the constant-weight embedding”. In Section 3.2, we express the constraints of non-binary SPC codes using these embeddings and then show properties of the constraints. In Section 4, we characterize the properties of the relaxed constraints. In particular, we introduce a rotations step that normalizes the geometry under consideration such that we only need to study the geometry induced by the SPC code defined by the all-ones check.

#### 3.1 Embedding finite fields

One key component of LP decoding is in building a connection between the finite fields, in which the codes are defined, and the Euclidean space, in which optimization algorithms operates. In [2], the mapping  $\mathbb{F}_2 \mapsto \mathbb{R}$  is defined by  $0 \rightarrow 0$  and  $1 \rightarrow 1$ . This straightforward transformation cannot be applied to the ML decoding problem of non-binary linear codes, where codes live in  $\mathbb{F}_q^N$ . We focus on embedding methods in this section and defer our discussions of the LP decoding formulation to Section 5.

In [16], Flanagan *et al.* introduce a mapping that embeds elements in  $\mathbb{F}_q$  into the Euclidean space of dimension  $q - 1$ .

**Definition 3** (*Flanagan's embedding [16]*) Let  $\mathbf{f} : \mathbb{F}_q \mapsto \{0, 1\}^{q-1}$  be a mapping such that for  $\alpha \in \mathbb{F}_q$ ,

$$\mathbf{f}(\alpha) := \mathbf{x} = (x_1, x_2, \dots, x_{q-1})$$

where

$$x_\delta = \begin{cases} 1, & \text{if } \delta = \alpha, \\ 0, & \text{if } \delta \neq \alpha. \end{cases}$$

We note that  $\mathbf{f}(\alpha)$  is a vector of length  $q - 1$  with at most one 1. Using this mapping,  $0 \in \mathbb{F}_q$  is mapped to the all-zeros vector of length  $q - 1$ , which has Hamming weight 0. On the other hand, all other elements of  $\mathbb{F}_q$  are mapped to binary vectors of Hamming weight 1.

In a more recent work [20], the authors use a slightly different embedding:

**Definition 4** (*constant-weight embedding [20]*) Let  $\mathbf{f}' : \mathbb{F}_q \mapsto \{0, 1\}^q$  be a mapping such that for  $\alpha \in \mathbb{F}_q$ ,

$$\mathbf{f}'(\alpha) := \mathbf{x} = (x_0, x_1, \dots, x_{q-1})$$

where

$$x_\delta = \begin{cases} 1, & \text{if } \delta = \alpha, \\ 0, & \text{if } \delta \neq \alpha. \end{cases}$$

Now  $\mathbf{f}'(\alpha)$  is a vector of length  $q$ . In addition, the Hamming weight of  $\mathbf{f}'(\alpha)$  is 1 for all values of  $\alpha \in \mathbb{F}_q$ . For this reason, we name this embedding method *constant-weight embedding*.

Using Flanagan's embedding in Definition 3, a vector  $\mathbf{c} \in \mathbb{F}_q^d$  can be mapped to a length  $(q - 1)d$  vector  $\mathbf{F}_v(\mathbf{c}) = (\mathbf{f}(c_1) | \mathbf{f}(c_2) | \dots | \mathbf{f}(c_n))^T$ . We can also write the above vector in an equivalent matrix of dimension  $(q - 1) \times d$  as:  $\mathbf{F}(\mathbf{c}) = (\mathbf{f}(c_1)^T | \mathbf{f}(c_2)^T | \dots | \mathbf{f}(c_n)^T)$ . In this representation, each column represents a coordinate of vector  $\mathbf{c}$ . We use the term *embedded matrix* to refer to the matrix-form embedding of a codeword. Further, let  $\mathbf{F}(\mathcal{C})$  and  $\mathbf{F}_v(\mathcal{C})$  be images of set  $\mathcal{C}$  under the two mappings respectively. That is,  $\mathbf{F}(\mathcal{C}) = \{\mathbf{y} | \mathbf{y} = \mathbf{F}(\mathbf{c}) \text{ for } \mathbf{c} \in \mathcal{C}\}$  and  $\mathbf{F}_v(\mathcal{C}) = \{\mathbf{y} | \mathbf{y} = \mathbf{F}_v(\mathbf{c}) \text{ for } \mathbf{c} \in \mathcal{C}\}$ . Similar definitions also apply to the constant-weight embedding in Definition 4. Further, we use  $\mathbf{F}'(\cdot)$  and  $\mathbf{F}'_v(\cdot)$  to denote the respective operations of embedding a vector using matrices and vectors following the constant-weight embedding method. Following the convention of Definition 4, we index rows of the matrix obtained by  $\mathbf{F}'(\cdot)$  from zero.

The two embedding methods have many aspects in common. In particular, many definitions and lemmas for one embedding method can easily be extended to the other embedding. We show in Section 5.2 that both embeddings achieve the same WER for LP decoding. On the other hand, there are two important differences between the two embeddings that make each useful for particular purposes. First, Flanagan's embedding saves exactly 1 coordinate for each embedded vector which typically translates into lower complexity. In particular, we show in Section 7 that ADMM LP decoding using the constant-weight embedding is slower than when Flanagan's embedding is used. Second, the constant-weight embedding is symmetric to all elements in the field while Flanagan's embedding treats 0 differently from other elements. This symmetry makes the constant-weight embedding useful when trying to solve non-convex programs, such as the penalized decoder described in Section 6.

### 3.2 Embedding of single parity-check codes in $\mathbb{F}_{2^m}$

We now restrict our discussion to fields of characteristic two. We further focus on Flanagan's embedding for conciseness. It is easy to extend all the definitions and lemmas to the constant-weight embedding. These extensions are presented in Appendix 9.

A  $d_c$ -dimensional<sup>4</sup> SPC code in  $\mathbb{F}_{2^m}$  is defined as

$$\mathcal{C} = \left\{ \mathbf{c} \in \mathbb{F}_{2^m}^{d_c} \mid \sum_{j=1}^{d_c} c_j h_j = 0 \right\}, \quad (3.1)$$

where the addition is in  $\mathbb{F}_{2^m}$  and where  $\mathbf{h} = (h_1, h_2, \dots, h_{d_c}) \in \mathbb{F}_{2^m}^{d_c}$  is the check vector. We are interested in characterizing  $F(\mathcal{C})$  because it arises in the relaxation of LP decoding. In [20] Honda and Yamamoto use a set of constraints derived from (3.1) that tests whether an embedding is in  $F(\mathcal{C})$  to introduce their relaxation. However, the authors did not formally study these constraints (on SPC code embeddings). In this paper, we make the following contributions: First, we characterize  $F(\mathcal{C})$  and  $F'(\mathcal{C})$ , i.e., SPC code embeddings for both embedding methods. Second, we explicitly state a factor graph representation of the embedding of the SPC code and derive properties of the factor graph. Third, we discuss the redundancy of these constraints. Finally, we characterize  $F(\mathcal{C})$  for  $\mathbb{F}_{2^2}$  in Appendix 12.

To begin with, we first formally define the binary vector representation of  $\mathbb{F}_{2^m}$  that appeared in Example 2:

**Definition 5** For an element  $\alpha \in \mathbb{F}_{2^m}$ , let

$$\mathbf{b}(\alpha) = (b_1, b_2, \dots, b_m)$$

be the binary vector representation of  $\alpha$ . In other words, the polynomial representation for  $\alpha$  is

$$p(x) = b_m x^{m-1} + b_{m-1} x^{m-2} + \dots + b_1.$$

Then, for any vector  $\mathbf{c} \in \mathbb{F}_{2^m}^{d_c}$ , the “bit matrix” representation of  $\mathbf{c}$  is a binary matrix of dimension  $m \times d_c$ . Denote this representation as

$$\mathbf{B}(\mathbf{c}) = (\mathbf{b}(c_1)^T \mid \mathbf{b}(c_2)^T \mid \dots \mid \mathbf{b}(c_{d_c})^T).$$

**Example 6** In  $\mathbb{F}_{2^3}$ ,  $\mathbf{B}((1, 2, 3)) = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$ .

Consider the set of embeddings defined in Definition 7 when Flanagan's embedding is used. We show in Lemma 8 that Definition 7 characterizes  $F(\mathcal{C})$ .

**Definition 7** Under Flanagan's embedding let  $q = 2^m$  and denote by  $\mathcal{E}$  the set of valid embedded matrices defined by a length- $d_c$  check  $\mathbf{h}$  where any  $(q-1) \times d_c$  binary matrix  $\mathbf{F} \in \mathcal{E}$  if and only if it satisfies the following conditions:

- (a)  $f_{ij} \in \{0, 1\}$ , where  $f_{ij}$  denotes the entry in the  $i$ -th row and  $j$ -th column of matrix  $\mathbf{F}$ .

---

<sup>4</sup>We use  $d_c$  to denote the length of the code because it is related to the check degree of an LDPC code.

(b)  $\sum_{i=1}^{q-1} f_{ij} \leq 1$  where  $q = 2^m$ .

(c) For any non-zero  $h \in \mathbb{F}_{2^m}$  and  $k \in [m]$ , let

$$\mathcal{B}(k, h) := \{\alpha | \mathbf{b}(h\alpha)_k = 1, \alpha \in \mathbb{F}_{2^m}\}, \quad (3.2)$$

where  $\cdot_k$  denotes the  $k$ -th entry of the vector. Let  $g_j^k = \sum_{i \in \mathcal{B}(k, h_j)} f_{ij}$ ,<sup>5</sup> then

$$\sum_{j=1}^{d_c} g_j^k = 0 \quad \text{for all } k \in [m] \quad (3.3)$$

where the addition is in  $\mathbb{F}_2$ .

**Lemma 8** In  $\mathbb{F}_{2^m}$ , let  $\mathcal{C}$  be the set of codewords that correspond to check  $\mathbf{h} = (h_1, h_2, \dots, h_{d_c})$ . Then

$$F(\mathcal{C}) = \mathcal{E}, \quad (3.4)$$

where  $\mathcal{E}$  is defined by Definition 7.

**Proof** See Appendix 10.2. ■

**Proposition 9** In  $\mathbb{F}_{2^m}$ , for all  $k \in [m]$  and  $h \neq 0$ ,

$$|\mathcal{B}(k, h)| = 2^{m-1}. \quad (3.5)$$

**Proof** See Appendix 10.1. ■

We can obtain a similar definition for the constant-weight embedding by changing two statements in Definition 7. First, the dimension of valid embedded matrices become  $q \times d_c$ . Second, condition (b) from Definition 7 becomes  $\sum_{i=0}^{q-1} f_{ij} = 1$ . Note that here the vector index starts from 0. Using this definition, a similar statement holds for the constant-weight embedding (cf. Lemma 35). In Appendix 9 we show the equivalent definition and lemma for the the constant-weight embedding.

**Definition 10** The conditions in Definition 7 can be represented using a factor graph. In this graph, there are three types of nodes: (i) variable nodes, (ii) parity-check nodes and (iii) at-most-one-on-check node. Each variable node corresponds to an entry  $f_{ij}$  for  $1 \leq i \leq 2^m - 1$  and  $1 \leq j \leq d_c$ . Each parity-check node corresponds to a parity-check of the  $k$ -th bit where  $k \in [m]$ ; it connects all variable nodes in  $\mathcal{B}(k, h_j)$  for all  $j \in [d_c]$ . Each at-most-one-on-check node corresponds to a constraint  $\sum_{i=1}^{q-1} f_{ij} \leq 1$ . We use circles to represent variables nodes and squares to represent parity-check nodes. In addition, we use triangles to represent at-most-one-on-check nodes because the polytope we get by relaxing this constraint is  $\mathbb{S}_{q-1}$ .

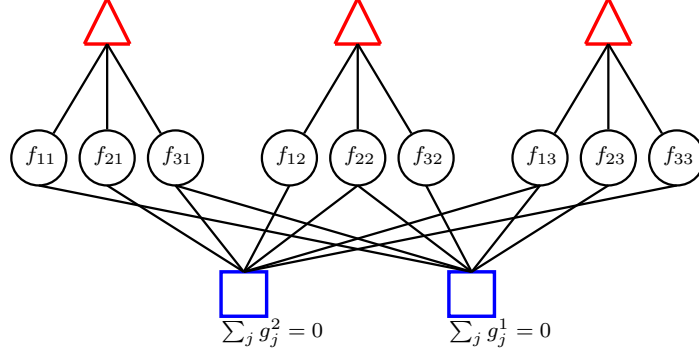
**Example 11** Consider  $\mathbb{F}_{2^2}$  and let  $\mathbf{h} = (1, 2, 3)$ . Let  $\mathcal{C}$  be the SPC code defined by  $\mathbf{h}$ . For a word  $\mathbf{c} \in \mathbb{F}_{2^2}^3$ , denote by

$$\mathbf{F} := F(\mathbf{c}) = \begin{pmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{pmatrix},$$

---

<sup>5</sup>Note that the sum is in the real space but can only be 1 or 0. Thus  $g_j^k$  is still in  $\mathbb{F}_2$ .





**Figure 1:** The factor graph for valid embeddings defined by check  $(1, 2, 3) \in \mathbb{F}_2^3$ . Each square represents a parity-check node; each triangle represents an at-most-one-on-check node.

and denote by

$$\mathbf{B} := \mathbf{B}(\mathbf{c}) = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{pmatrix}.$$

By Lemma 8,  $\mathbf{c} \in \mathcal{C}$  if and only if  $\mathbf{F}$  satisfies the following conditions:

- (a)  $f_{ij} \in \{0, 1\}$ .
- (b)  $\sum_{i=1}^3 f_{ij} \leq 1$ , where addition is in  $\mathbb{R}$ .
- (c) We can calculate from Table 2 that  $\mathcal{B}(1, 1) = \{1, 3\}$ ,  $\mathcal{B}(2, 1) = \{2, 3\}$ ,  $\mathcal{B}(1, 2) = \{2, 3\}$ ,  $\mathcal{B}(2, 2) = \{1, 2\}$ ,  $\mathcal{B}(1, 3) = \{1, 3\}$ , and  $\mathcal{B}(2, 3) = \{1, 3\}$ . (Cf. (3.2) for the definition of  $\mathcal{B}(k, h)$ ). Therefore by Definition 7,  $\mathbf{g}^2 = (f_{21} + f_{31}, f_{12} + f_{22}, f_{13} + f_{33})$  and  $\sum_j g_j^2 = 0$  (in  $\mathbb{F}_2$ ). Similarly,  $\mathbf{g}^1 = (f_{11} + f_{31}, f_{22} + f_{32}, f_{13} + f_{23})$  and  $\sum_j g_j^1 = 0$  (in  $\mathbb{F}_2$ ).

Using Definition 10, we draw the factor graph in Figure 1. Each square represents a parity-check and each triangle represents an at-most-one-on-check.

One important remark regarding Definition 7 is that it is not the unique way of defining valid embeddings. In fact in [20], the authors' relaxation is implicitly based on the conditions in Lemma 12 below.

**Lemma 12** Consider Flanagan's embedding and let  $\mathcal{F}$  be the set of  $(q - 1) \times d_c$  binary matrices defined by the first two conditions of Definition 7 but with the third condition replaced by the condition  $(c^*)$  defined below, then  $\mathcal{F} = \mathcal{E}$ . The complete set of conditions are

- (a)  $f_{ij} \in \{0, 1\}$ , where  $f_{ij}$  denotes the entry in the  $i$ -th row and  $j$ -th column of matrix  $\mathbf{F}$ .
- (b)  $\sum_{i=1}^{q-1} f_{ij} \leq 1$  where  $q = 2^m$ .
- (c\*) Let  $\mathcal{K}$  be any nonempty subset of  $[m]$ . For any non-zero  $h \in \mathbb{F}_{2^m}$ , let  $\tilde{\mathcal{B}}(\mathcal{K}, h) := \{\alpha \mid \sum_{k \in \mathcal{K}} \mathbf{b}(h\alpha)_k = 1\}$ , where  $\mathbf{b}(h\alpha)_k$  denotes the  $k$ -th entry of the vector  $\mathbf{b}(h\alpha)$ . Let  $g_j^{\mathcal{K}} = \sum_{i \in \tilde{\mathcal{B}}(\mathcal{K}, h_j)} f_{ij}$ , then  $\sum_{j=1}^{d_c} g_j^{\mathcal{K}} = 0$  for all  $\mathcal{K} \subset [m]$ ,  $\mathcal{K} \neq \emptyset$ , where the addition is in  $\mathbb{F}_2$ .

**Proof** See Appendix 10.3 ■

**Lemma 13** In  $\mathbb{F}_{2^m}$ , for all  $\mathcal{K} \subset [m]$  and  $h \neq 0$ ,

$$|\tilde{\mathcal{B}}(\mathcal{K}, h)| = 2^{m-1}.$$

**Proof** See Appendix 10.4. ■

### Remarks

Note that many redundant constraints are used to define  $\mathcal{F}$  in Lemma 12. Similar to redundant parity-checks for binary linear codes (see e.g., [22]), these redundant constraints can be used to tighten the polytope to be discussed in Section 4. On the other hand, adding such extra constraints generally increases computational complexity. Especially, note that there are  $m$  parity-checks in Definition 7 whereas the number of parity-checks in Lemma 12 is  $2^{m-1}$ .

## 4 Geometry of the relaxation of embeddings

In this section, we consider embeddings of non-binary SPC codes and study the polytopes obtained from these embeddings. There are two ways of obtaining polytopes relevant to LP decoding. In [16], the authors study the convex hull of  $\mathbf{F}(\mathcal{C})$ . However, this polytope is not easy to characterize. On the other hand, using the factor graph representation of SPC code embeddings from the previous section we can get a relaxation of  $\text{conv}(\mathbf{F}(\mathcal{C}))$  following the methodology of LP decoding of binary LDPC codes in [2]. In other words, the factor graph representation of embeddings resembles a mini binary (“generalized”) LDPC code. Instead of taking the convex hull of all codewords, we can take the intersection of many local polytopes for simplicity. This idea is first seen in [20] for the constant-weight embedding. We first restate the relaxation method in [20] for Flanagan’s embedding; readers are referred to Appendix 9 for the corresponding definitions when the constant-weight embedding is used. We note that this restatement is important because it allows us to leverage off-the-shelf techniques to develop an efficient algorithm. We then show a tightness result of the parity polytope, which can imply a tightness result on the polytope of SPC code embeddings. Finally, we introduce several rotation properties of the polytope of SPC code embeddings that are useful for developing an efficient decoding algorithm.

### 4.1 Relaxation of single parity-check code embeddings

We first describe the relaxation considered in [16].

**Definition 14** Let  $\mathcal{C}$  be a non-binary SPC code defined by check vector  $\mathbf{h}$ . Denote by  $\mathbb{V}$  the “tight code polytope” for Flanagan’s embedding where

$$\mathbb{V} = \text{conv}(\mathbf{F}(\mathcal{C})).$$

The following definition extends the relaxation in [20] to Flanagan’s embedding.

**Definition 15** Let  $\mathcal{C}$  be a non-binary SPC code defined by check vector  $\mathbf{h}$ . In  $\mathbb{F}_{2^m}$  denote by  $\mathbb{U}$  the “relaxed code polytope”<sup>6</sup> for Flanagan’s embedding where a  $(q-1) \times d_c$  matrix  $\mathbf{F} \in \mathbb{U}$  if and only if the following constraints hold:

---

<sup>6</sup>From the visual appearances of the letters, one can think of  $\mathbb{U}$  as a relaxation of  $\mathbb{V}$ .

- (a)  $f_{ij} \in [0, 1]$ .
- (b)  $\sum_{i=1}^{q-1} f_{ij} \leq 1$ .
- (c) Let  $\tilde{\mathcal{B}}(\mathcal{K}, h)$  be the same set defined in Lemma 12. Let  $\mathbf{g}^{\mathcal{K}}$  be a vector of length  $d_c$  such that  $g_j^{\mathcal{K}} = \sum_{i \in \tilde{\mathcal{B}}(\mathcal{K}, h_j)} f_{ij}$ , then

$$\mathbf{g}^{\mathcal{K}} \in \mathbb{P}_{d_c} \quad \text{for all } \mathcal{K} \subset [m] \text{ and } \mathcal{K} \neq \emptyset \quad (4.1)$$

We note that in Definition 15 conditions (a) and (b) define the simplex  $\mathbb{S}_{q-1}$ . In addition, condition (c) leverages parity polytope  $\mathbb{P}_{d_c}$ . Both polytopes are studied extensively in the literature (e.g. [12] and [23]). We show how to leverage this definition in developing efficient decoding algorithms in Section 5.

## 4.2 A tightness result for the relaxed code polytope

We note that Definition 15 is not the only way to relax the constraints in Lemma 12. In this subsection we study another possible relaxation and show that the relaxation in Definition 15 is tighter.

Consider  $\mathbf{h} = (1, 2, 3) \in \mathbb{F}_2^3$  as a motivating example. Lemma 12 requires that  $\mathbf{g}^{\{2\}}$  has even parity ( $\mathcal{K} = \{1\}$  in this case). This means  $(f_{21} + f_{31}) + (f_{12} + f_{22}) + (f_{13} + f_{33}) = 0$  in  $\mathbb{F}_2$  (cf. Example 11). Now consider the following two possible relaxations. We let  $0 \leq f_{ij} \leq 1$  and  $\sum_i f_{ij} \leq 1$  for both cases.

- (i) Vector  $(f_{21}, f_{31}, f_{22}, f_{32}, f_{23}, f_{33}) \in \mathbb{P}_6$ .
- (ii) Vector  $(f_{21} + f_{31}, f_{22} + f_{32}, f_{23} + f_{33}) \in \mathbb{P}_3$  (the same relaxation used in Definition 15).

One would argue that both relaxations are applicable to LP decoding because the integral points of both relaxations are embeddings that satisfy Lemma 12. We justify using Proposition 16 that statement (ii) implies statement (i) for more general cases. Thus, Definition 15 is a better relaxation method.

**Proposition 16** *Let*

$$\mathbf{x} := (\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^s)$$

*be a non-negative vector obtained by concatenating  $s$  row vectors  $\mathbf{x}^i$ , where each  $\mathbf{x}^i$  is a length- $t$  vector. Let  $\mathbf{y} := (\|\mathbf{x}^1\|_1, \|\mathbf{x}^2\|_1, \dots, \|\mathbf{x}^s\|_1)$ . Then  $\mathbf{y} \in \mathbb{P}_s$  implies  $\mathbf{x} \in \mathbb{P}_{st}$ .*

**Proof** See Appendix 10.5. ■

Note that the converse of Proposition 16 does not hold. That is, if  $\mathbf{x} \in \mathbb{P}_{st}$  and  $\|\mathbf{x}^i\|_1 \leq 1$  for all  $i$   $\mathbf{y}$  does not necessarily belong to  $\mathbb{P}_s$ . As a counter example, let  $\mathbf{x} = (0.5, 0.5, 0, 0, 0, 0)$ . Then  $\mathbf{x} \in \mathbb{P}_6$  but  $\mathbf{y} = (1, 0, 0) \notin \mathbb{P}_3$ .

By applying Proposition 16 to our example, we conclude that  $(f_{21} + f_{31}, f_{22} + f_{32}, f_{23} + f_{33}) \in \mathbb{P}_3$  implies that  $(f_{21}, f_{31}, f_{22}, f_{32}, f_{23}, f_{33}) \in \mathbb{P}_6$ .

### 4.3 Rotation

We describe a rotation operation that normalizes the geometry under consideration. This process simplifies our study of geometries because one only needs to work on a canonical object instead of different polytopes for different checks. A similar idea is also found in BP decoding in [17].

As a side effect, the normalization results in a column permutation property for valid embedded matrices. In particular, post-normalization, any permutation when applied to columns of a valid SPC code embedded matrix is a valid SPC code embedded matrix. Note that the convex hull of all permutations of a vector can be defined using (“single-variate” a.k.a., “vector”) majorization [24]. This observation is the key to the algorithm of projection onto the parity polytope [12]. Similarly, the convex hull of all column permutations of a matrix can be defined using multivariate majorization [24]. Thus it may be possible to use results for multivariate majorization to characterize  $\text{conv}(\mathcal{F}(\mathcal{C}))$ . We leave this as future work.

We first describe the rotation operation for  $\text{conv}(\mathcal{F}(\mathcal{C}))$ . We then show that the same operation can be applied to the relaxed code polytope  $\mathbb{U}$ .

#### 4.3.1 Rotation for the tight code polytope

Consider an arbitrary finite field  $\mathbb{F}_q$  and let  $\mathcal{C}^{\mathbf{N}}$  be the “normalized” set of codewords defined as  $\mathcal{C}^{\mathbf{N}} := \{\mathbf{x} \mid \sum_{j=1}^{d_c} x_j = 0\}$ . We focus on Flanagan’s embedding and let  $\mathcal{E}^{\mathbf{N}} := \mathcal{F}(\mathcal{C}^{\mathbf{N}})$ , where the addition is in  $\mathbb{F}_{2^m}$ . Then for any check that is not “normalized” (i.e., contains entries not equal to 1), we can obtain the set of codewords  $\mathcal{C}$  from  $\mathcal{C}^{\mathbf{N}}$  by dividing each entry by the corresponding check value in  $\mathbb{F}_q$ . Formally, let  $\mathbf{h} = (h_1, \dots, h_{d_c})$  be the check vector and  $h_j \neq 0$ . Then

$$\mathcal{C} = \{\mathbf{y} \mid y_j = x_j/h_j \text{ for all } j; \mathbf{x} \in \mathcal{C}^{\mathbf{N}}\}.$$

Note that in the equation above  $x_j \neq 0$  and  $y_j \neq 0$ . We define the rotation matrix  $\mathbf{D}$  as follows:

**Definition 17** Let  $h$  be a non-zero element in  $\mathbb{F}_q$ . Let  $\mathbf{D}(q, h)$  be a  $(q-1) \times (q-1)$  permutation matrix satisfying the following equation

$$\mathbf{D}(q, h)_{ij} = \begin{cases} 1 & \text{if } i \cdot h = j, \\ 0 & \text{otherwise,} \end{cases}$$

where the multiplication is in  $\mathbb{F}_q$ . For a vector  $\mathbf{h} \in \mathbb{F}_q^{d_c}$  that  $h_j \neq 0$  for all  $j \in [d_c]$ , let

$$\mathbf{D}(q, \mathbf{h}) = \text{diag}(\mathbf{D}(q, h_1), \dots, \mathbf{D}(q, h_{d_c})).$$

Using the permutation matrix  $\mathbf{D}(q, \mathbf{h})$ , we can obtain  $\mathcal{E}$  by rotating every vector in  $\mathcal{E}^{\mathbf{N}}$ :

$$\mathcal{E} := \{\mathbf{f} \mid \mathbf{f} = \mathbf{D}(q, \mathbf{h})\mathbf{e}, \mathbf{e} \in \mathcal{E}^{\mathbf{N}}\}.$$

**Example 18** Let  $q = 5$  and  $h = 3$ . Using division in  $\mathbb{F}_5$ , we obtain  $1 \cdot 3^{-1} = 2$ ,  $2 \cdot 3^{-1} = 4$ ,  $3 \cdot 3^{-1} = 1$  and  $4 \cdot 3^{-1} = 3$ . Therefore

$$\mathbf{D}(5, 3) = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}.$$

Take the equation  $1 \cdot 3^{-1} = 2$  for example. The respective embedded vectors of 1 and 2 are  $(1, 0, 0, 0)$  and  $(0, 1, 0, 0)$ . It is easy to verify that  $(0, 1, 0, 0)^T = \mathbf{D}(5, 3)(1, 0, 0, 0)^T$ .

We use  $\mathbb{V}^N$  to denote the tight code polytope defined by the all-ones check. The following lemma shows that we can obtain  $\mathbb{V}^N$  by rotating  $\mathbb{V}$  using  $\mathbf{D}(q, \mathbf{h})$ .

**Lemma 19** *Let  $\mathbf{h}$  be a check such that all entries of  $\mathbf{h}$  are non-zero. Let  $\mathbb{V}$  be the convex hull of  $\mathcal{E}$  and  $\mathbb{V}^N$  be the convex hull of  $\mathcal{E}^N$ . Then a vector  $\mathbf{u} \in \mathbb{V}$  if and only if  $\mathbf{D}(q, \mathbf{h})^{-1}\mathbf{u} \in \mathbb{V}^N$ . In other words, a vector  $\mathbf{w} \in \mathbb{V}^N$  if and only if  $\mathbf{D}(q, \mathbf{h})\mathbf{w} \in \mathbb{V}$ .*

**Proof** See Appendix 10.6. ■

This lemma implies that having different check values is nothing but a rotation of the normalized code polytope. The following lemma states that projections onto  $\mathbb{V}$  for arbitrary checks can be performed via projections onto  $\mathbb{V}^N$ .

**Lemma 20** *Let  $\mathbf{h}$  be a check such that all entries of  $\mathbf{h}$  are non-zero. Let  $\mathbb{V}$  be the convex hull of  $\mathcal{E}$  and let  $\mathbb{V}^N$  be the convex hull of  $\mathcal{E}^N$ . Let  $\mathbf{v}$  be any vector and let  $\mathbf{v}^N := \mathbf{D}(q, \mathbf{h})^{-1}\mathbf{v}$ . Then, if  $\mathbf{u}^N$  is the projection of  $\mathbf{v}^N$  onto  $\mathbb{V}^N$ , then  $\mathbf{u} := \mathbf{D}(q, \mathbf{h})\mathbf{u}^N$  is the projection of  $\mathbf{v}$  onto  $\mathbb{V}$ .*

**Proof** See Appendix 10.6. ■

Lemma 20 shows that we can obtain the projection of a vector  $\mathbf{v}$  onto any tight code polytope  $\mathbb{V}$  in three steps: First rotate the vector and get  $\mathbf{v}^N$ , then project  $\mathbf{v}^N$  onto the normalized polytope  $\mathbb{V}^N$ , and finally rotate the projection result  $\mathbf{v}^N$  back to the original coordinate system.

#### 4.3.2 Rotation for the relaxed code polytope

In  $\mathbb{F}_{2^m}$ , we can relax  $\mathbb{V}$  using Definition 15 and obtain a relaxed code polytope  $\mathbb{U}$ . The same permutation matrix  $\mathbf{D}(q, \mathbf{h})$  can also be applied to  $\mathbb{U}$ .

**Lemma 21** *Let  $\mathbf{h}$  be a check such that all entries of  $\mathbf{h}$  are non-zero. Let  $\mathbb{U}$  be the relaxed code polytope obtained by Definition 15 for check  $\mathbf{h}$  and let  $\mathbb{U}^N$  be the relaxed code polytope obtained by Definition 15 for the all-ones check. Then a vector  $\mathbf{v} \in \mathbb{U}$  if and only if  $\mathbf{D}(q, \mathbf{h})^{-1}\mathbf{v} \in \mathbb{U}^N$ . In other words, a vector  $\mathbf{w} \in \mathbb{U}^N$  if and only if  $\mathbf{D}(q, \mathbf{h})\mathbf{w} \in \mathbb{U}$ .*

**Proof** See Appendix 10.6. ■

Similar to the previous case, we have the following lemma which applies to the projection operation.

**Lemma 22** *Let  $\mathbf{h}$  be a check such that all entries of  $\mathbf{h}$  are non-zero. Let  $\mathbb{U}$  be the relaxed code polytope obtained by Definition 15 for check  $\mathbf{h}$  and let  $\mathbb{U}^N$  be the relaxed code polytope obtained by Definition 15 for the all-one check. Let  $\mathbf{v}$  be any vector and let  $\mathbf{v}^N := \mathbf{D}(q, \mathbf{h})^{-1}\mathbf{v}$ . Then, if  $\mathbf{u}^N$  is the projection of  $\mathbf{v}^N$  onto  $\mathbb{U}^N$ , then  $\mathbf{u} := \mathbf{D}(q, \mathbf{h})\mathbf{u}^N$  is the projection of  $\mathbf{v}$  onto  $\mathbb{U}$ .*

**Proof** See Appendix 10.6. ■

#### 4.4 Remarks

For  $\mathbb{F}_{2^2}$ , we can obtain a characterization of SPC code embeddings simpler than Definition 7. In addition, we note that it is mentioned by [20] (and we paraphrase) that  $\mathbb{V} = \mathbb{U}$  for codes in  $\mathbb{F}_{2^2}$ . Yet no proof was given in [20]. We validated this statement numerically and believe that it holds. These results for  $\mathbb{F}_{2^2}$  are discussed in Appendix 12.

For  $\mathbb{F}_{2^m}$  where  $m \geq 3$ , it is obvious that  $\mathbb{V} \subset \mathbb{U}$ . However, we do not have evidence that  $\mathbb{V} = \mathbb{U}$ . In fact, simulation results in Section 7.1 indicate that  $\mathbb{V} \neq \mathbb{U}$  because the WER performance of LP decoding based on  $\mathbb{U}$  is worse than that based on  $\mathbb{V}$ . Understanding how loose  $\mathbb{U}$  is compared to  $\mathbb{V}$  is important future work.

## 5 LP decoding and ADMM formulations

LP decoding of non-binary codes was first introduced by Flanagan *et al.* in [16]. The LP decoding problem in [16] is based on the embedding method presented in Definition 3. Honda and Yamamoto proposed a different LP decoding problem in [20], one that uses the embedding method presented in Definition 4. However, no connection between the problem in [16] and the problem in [20] has been made. We first review these two LP decoding problems and show that the choice of embedding method does not affect the results of LP decoding. We then develop two ADMM algorithms for LP decoding. The first algorithm is generic and can be applied to arbitrary fields. However, it requires a sub-routine that can project either onto the tight code polytope or onto the relaxed code polytope. In our previous work [25], we proposed an ADMM projection algorithm to project onto the relaxed code polytope  $\mathbb{U}$ . The resulting LP decoding algorithm executes ADMM iterations such that each iteration contains  $M$  ADMM projection sub-routines, where  $M$  is the number of checks. In other words there are two levels of ADMM algorithms: A global ADMM decoding, and  $M$  ADMM sub-routines per iteration of that global ADMM decoding. We observe that this algorithm is not especially efficient because even if the projection sub-routines converge reasonably fast, the multiplicity of sub-routines is large ( $M$  per decoding iteration). To solve this problem, we introduce another ADMM formulation for codes in  $\mathbb{F}_{2^m}$  in Section 5.4 in which no ADMM sub-routine is required. This formulation leverages the factor graph representation of embeddings described in Section 3.2.

### 5.1 Review of LP decoding problems

We first review the LP decoding problem proposed by Flanagan *et al.* in [16]. We consider a linear code specified by a parity-check matrix  $\mathbf{H}$ , where each entry  $h_{ji} \in \mathbb{F}_q$ . Let  $\Sigma$  be the output space of the channel. In [16], a function  $\mathbf{L} : \Sigma \mapsto (\mathbb{R} \cup \{\pm\infty\})^{q-1}$  is introduced, it is defined as

$$\mathbf{L}(y) := \boldsymbol{\lambda} = (\lambda_1, \dots, \lambda_{q-1}),$$

where for each  $y \in \Sigma$ ,  $\delta \in \mathbb{F}_q \setminus \{0\}$ ,  $\lambda_\delta = \log \left( \frac{\Pr[Y=y|X=0]}{\Pr[Y=y|X=\delta]} \right)$ . Similar to  $F_v(\cdot)$  defined in Section 3.1, we define  $\mathbf{L}_v(\mathbf{y}) = (\mathbf{L}(y_1)|\mathbf{L}(y_2)|\dots|\mathbf{L}(y_n))^T$ . Maximum likelihood decoding can then be rewritten as (cf. [16])

$$\begin{aligned} \hat{\mathbf{c}} &= \operatorname{argmin}_{\mathbf{c} \in \mathcal{C}} \sum_{i=1}^n \log \left( \frac{\Pr[y_i|0]}{\Pr[y_i|c_i]} \right) \\ &= \operatorname{argmin}_{\mathbf{c} \in \mathcal{C}} \sum_{i=1}^n \mathbf{L}(y_i) \mathbf{f}(c_i)^T \\ &= \operatorname{argmin}_{\mathbf{c} \in \mathcal{C}} \mathbf{L}_v(\mathbf{y}) \mathbf{F}_v(\mathbf{c})^T. \end{aligned}$$

In [16], ML decoding is relaxed to a linear program. We can write the LP decoding problem (using the “tight” “Flanagan” representation,  $\mathbf{FT}$ ) as

$$\begin{aligned} \mathbf{LP}\text{-}\mathbf{FT}: \quad & \min \quad \mathbf{L}_v(\mathbf{y})^T \mathbf{x} \\ & \text{subject to} \quad \mathbf{P}_j \mathbf{x} \in \mathbb{V}_j, \forall j \in \mathcal{J}, \end{aligned} \tag{5.1}$$

where  $\mathbf{P}_j$  selects the variables that participate in the  $j$ -th check and  $\mathbb{V}_j = \operatorname{conv}(F_v(\mathcal{C}_j))$ . Here  $\mathcal{C}_j$  is the SPC code defined by the non-zero entries of the  $j$ -th check.

Using the relaxation of Definition 15, we can formulate a different and further relaxed LP decoding problem. Note that this problem only applies to codes in  $\mathbb{F}_{2^m}$ . We write this (“Flanagan” “relaxed”, **FR**) problem as

$$\begin{aligned} \textbf{LP-FR:} \quad & \min \quad \mathbf{L}_v(\mathbf{y})^T \mathbf{x} \\ & \text{subject to} \quad \mathbf{P}_j \mathbf{x} \in \mathbb{U}_j, \forall j \in \mathcal{J}, \end{aligned} \quad (5.2)$$

where  $\mathbf{P}_j$  is the same as the previous problem and  $\mathbb{U}_j$  is defined by Definition 15 for the  $j$ -th check.

Since the LP decoding problem in [20] uses the constant-weight embedding, both the objective function and the constraint set are different from those in [16]. Let

$$\mathbf{L}'(y) := \boldsymbol{\lambda} = (\lambda_0, \dots, \lambda_{q-1}),$$

where for each  $y \in \Sigma$ ,  $\delta \in \mathbb{F}_q$ ,  $\lambda_\delta = \log \left( \frac{1}{\Pr[Y=y|X=\delta]} \right)$ . The LP decoding problem (“constant-weight” “relaxed”, **CR**) in [20] is

$$\begin{aligned} \textbf{LP-CR:} \quad & \min \quad \mathbf{L}'_v(\mathbf{y}) \mathbf{x} \\ & \text{subject to} \quad \mathbf{P}'_j \mathbf{x} \in \mathbb{U}'_j, \forall j \in \mathcal{J}, \end{aligned} \quad (5.3)$$

where  $\mathbf{P}'_j$  selects the variables that participate in the  $j$ -th check and  $\mathbb{U}'_j$  is the relaxed code polytope when the constant-weight embedding is used (cf. Definition 38). We can formulate a different, “constant-weight”, “tighter” (**CT**) LP decoding problem using  $\mathbb{V}'_j := \text{conv}(\mathbf{F}'_v(\mathcal{C}_j))$  as

$$\begin{aligned} \textbf{LP-CT:} \quad & \min \quad \mathbf{L}'_v(\mathbf{y})^T \mathbf{x} \\ & \text{subject to} \quad \mathbf{P}'_j \mathbf{x} \in \mathbb{V}'_j, \forall j \in \mathcal{J}. \end{aligned} \quad (5.4)$$

**Example 23** In this example we demonstrate the selection matrices  $\mathbf{P}$  and  $\mathbf{P}'$ . Consider a code in  $\mathbb{F}_{2^2}$  defined by the following parity-check matrix:  $\mathbf{H} = \begin{pmatrix} 1 & 2 & 2 & 3 \\ 2 & 0 & 1 & 2 \end{pmatrix}$ . Then  $\mathbf{P}_1 = \mathbf{I}_{12 \times 12}$  and

$$\mathbf{P}_2 = \begin{pmatrix} \mathbf{I}_{3 \times 3} & \mathbf{0}_{3 \times 3} & \mathbf{0}_{3 \times 3} & \mathbf{0}_{3 \times 3} \\ \mathbf{0}_{3 \times 3} & \mathbf{0}_{3 \times 3} & \mathbf{I}_{3 \times 3} & \mathbf{0}_{3 \times 3} \\ \mathbf{0}_{3 \times 3} & \mathbf{0}_{3 \times 3} & \mathbf{0}_{3 \times 3} & \mathbf{I}_{3 \times 3} \end{pmatrix}. \quad \mathbf{P}'_1 = \mathbf{I}_{16 \times 16} \text{ and } \mathbf{P}'_2 = \begin{pmatrix} \mathbf{I}_{4 \times 4} & \mathbf{0}_{4 \times 4} & \mathbf{0}_{4 \times 4} & \mathbf{0}_{4 \times 4} \\ \mathbf{0}_{4 \times 4} & \mathbf{0}_{4 \times 4} & \mathbf{I}_{4 \times 4} & \mathbf{0}_{4 \times 4} \\ \mathbf{0}_{4 \times 4} & \mathbf{0}_{4 \times 4} & \mathbf{0}_{4 \times 4} & \mathbf{I}_{4 \times 4} \end{pmatrix}.$$

## 5.2 Equivalence properties between two embedding methods

In this section, we show several equivalence properties between Flanagan’s embedding and the constant-weight embedding in the context of LP decoding. As a reminder, we consider two LP decoding objectives and two types of LP decoding constraints. Therefore there are four LP decoding problems under consideration. We summarize the notation in Table 3.

	Tight code polytope	Relaxed code polytope
Flanagan Emb.	<b>LP-FT</b>	<b>LP-FR</b>
Const.-weight Emb.	<b>LP-CT</b>	<b>LP-CR</b>

**Table 3:** Four LP decoding problems

### 5.2.1 Equivalence between LP-FT and LP-CT

We first focus on LP decoding using the tight code polytope  $\mathbb{V}$ .

**Theorem 24** *Consider any finite field  $\mathbb{F}_q$  and let  $\mathbf{y} \in \Sigma$ . Let  $\mathbf{f}_i$  be a length- $(q-1)$  vector where  $i \in \mathcal{I}$ . If  $\hat{\mathbf{f}} = (\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_N)^T$  is the solution of **LP-FT**, then  $\bar{\mathbf{f}} = (1 - \|\mathbf{f}_1\|_1, \mathbf{f}_1, 1 - \|\mathbf{f}_2\|_1, \mathbf{f}_2, \dots, 1 - \|\mathbf{f}_N\|_1, \mathbf{f}_N)^T$  is the solution of **LP-CT**. Conversely, if  $\bar{\mathbf{f}} = (a_1, \mathbf{f}_1, a_2, \mathbf{f}_2, \dots, a_N, \mathbf{f}_N)^T$  is the solution of **LP-CT** for some  $a_1, \dots, a_N$ , then  $\hat{\mathbf{f}} = (\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_N)^T$  is the solution of **LP-FT**.*

**Proof** See Appendix 10.7 ■

This theorem builds the equivalence in the sense that, if **LP-FT** and **LP-CT** are given the same channel output vector, then there is a bijective mapping between their decoding results. It is then easy to show the following corollary.

**Corollary 25** *Both **LP-FT** and **LP-CT** achieve the same symbol-error-rate and word-error-rate.*

**Proof** By Theorem 24, **LP-FT** decodes a channel output  $\mathbf{y}$  to a fractional solution if and only if **LP-CT** decodes  $\mathbf{y}$  to a fractional solution. Thus the two decoders either both succeed or both fail. ■

**Corollary 26** *Under the channel symmetry condition in the sense of [16], the probability that **LP-CT** fails is independent of the codeword that was transmitted.*

**Proof** This is due to Theorem 5.1 in [16] and Corollary 25. ■

### 5.2.2 Equivalence between LP-FR and LP-CR

We now consider the LP decoding problem for codes in fields of characteristic two when the relaxed code polytope is used in LP decoding.

**Theorem 27** *Consider finite fields  $\mathbb{F}_{2^m}$  and let  $\mathbf{y} \in \Sigma$ . Let  $\mathbf{f}_i$  be a length- $(2^m - 1)$  vector where  $i \in \mathcal{I}$ . If  $\hat{\mathbf{f}} = (\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_N)^T$  is the solution of **LP-FR**, then  $\bar{\mathbf{f}} = (1 - \|\mathbf{f}_1\|_1, \mathbf{f}_1, 1 - \|\mathbf{f}_2\|_1, \mathbf{f}_2, \dots, 1 - \|\mathbf{f}_N\|_1, \mathbf{f}_N)^T$  is the solution of **LP-CR**. Conversely, if  $\bar{\mathbf{f}} = (a_1, \mathbf{f}_1, a_2, \mathbf{f}_2, \dots, a_N, \mathbf{f}_N)^T$  is the solution of **LP-CR** for some  $a_1, \dots, a_N$ , then  $\hat{\mathbf{f}} = (\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_N)^T$  is the solution of **LP-FR**.*

**Proof** See Appendix 10.8. ■

**Corollary 28** *Both **LP-FR** and **LP-CR** achieve the same symbol-error-rate and word-error-rate.*

**Corollary 29** *Under the channel symmetry condition in the sense of [16], the probability that **LP-FR** fails is independent of the codeword that was transmitted.*

**Proof** This is due to Theorem 2 in [20] and Theorem 27. ■

We also provide our direct proof of this corollary in Appendix 10.9. Our proof leverages the proof technique taken in [16]. In addition, we show a symmetry property of  $\mathbb{U}$ .



### 5.2.3 Remarks

- Our results imply that one has the freedom to choose either Flanagan's embedding or the constant-weight embedding for LP decoding. Both embedding methods yield the same error rates.
- We note that the relaxed versions of LP decoding (**LP-FR** and **LP-CR**) are in general worse than the tighter versions (**LP-FT** and **LP-CT**) in terms of error rates.
- LP decoding using Flanagan's embedding requires fewer variables. In addition, we show in Section 7 that the ADMM LP decoder in Section 5.4, which uses Flanagan's embedding, converges faster than its constant-weight embedding counterpart.

### 5.3 Generic ADMM formulations of the LP decoding problem

In this section, we derive a generic ADMM formulation for LP decoding of non-binary codes. We formulate the ADMM algorithm under the assumption that a sub-routine that projects a vector onto  $\mathbb{V}$  or  $\mathbb{U}$  is provided. We will use the formulation in this section to develop a penalized LP decoder in Section 6.

We use the shorthand notation  $\boldsymbol{\gamma} := \mathbf{L}_v(\mathbf{y})$ , and following the methodology of [12], cast the LP into a form solvable using ADMM. We introduce replicas  $\mathbf{z}_j$  for all  $j \in \mathcal{J}$ . Each  $\mathbf{z}_j$  is a length- $(q-1)d_c$  vector where  $d_c$  is the check degree. We then express (5.1) in the following, equivalent, form:

$$\begin{aligned} \min \quad & \boldsymbol{\gamma}^T \mathbf{x} \\ \text{subject to} \quad & \mathbf{P}_j \mathbf{x} = \mathbf{z}_j, \\ & \mathbf{z}_j \in \mathbb{V}_j, \text{ for all } j \in \mathcal{J}, \\ & \mathbf{x}_i \in \mathbb{S}_{q-1}, \text{ for all } i \in \mathcal{I}, \end{aligned} \tag{5.5}$$

where  $\mathbf{x}_i = (x_{(i-1)(q-1)+1}, x_{(i-1)(q-1)+2}, \dots, x_{i(q-1)})$  is the sub-vector selected from the  $i$ -th  $(q-1)$ -length block of  $\mathbf{x}$ . In other words,  $\mathbf{x}_i$  corresponds to the embedded vector of the  $i$ -th non-binary symbol.  $\mathbb{S}_{q-1}$  is the  $q-1$  simplex defined in (2.2).

The augmented Lagrangian is

$$\mathcal{L}_\mu(\mathbf{x}, \mathbf{z}, \boldsymbol{\lambda}) = \boldsymbol{\gamma}^T \mathbf{x} + \sum_{j \in \mathcal{J}} \boldsymbol{\lambda}_j^T (\mathbf{P}_j \mathbf{x} - \mathbf{z}_j) + \frac{\mu}{2} \sum_{j \in \mathcal{J}} \|\mathbf{P}_j \mathbf{x} - \mathbf{z}_j\|_2^2.$$

ADMM iteratively performs the following updates:

$$\mathbf{x}\text{-update: } \mathbf{x}^{k+1} = \arg\min_{\mathbf{x}} \mathcal{L}_\mu(\mathbf{x}, \mathbf{z}^k, \boldsymbol{\lambda}^k), \tag{5.6}$$

$$\mathbf{z}\text{-update: } \mathbf{z}^{k+1} = \arg\min_{\mathbf{z}} \mathcal{L}_\mu(\mathbf{x}^{k+1}, \mathbf{z}, \boldsymbol{\lambda}^k), \tag{5.7}$$

$$\boldsymbol{\lambda}\text{-update: } \boldsymbol{\lambda}_j^{k+1} = \boldsymbol{\lambda}_j^k + \mu (\mathbf{P}_j \mathbf{x}^{k+1} - \mathbf{z}_j^{k+1}). \tag{5.8}$$

In the  $\mathbf{x}$ -update we solve the following optimization problem:

$$\min_{\forall i \in \mathcal{I}, \mathbf{x}_i \in \mathbb{S}_{q-1}} \mathcal{L}_\mu(\mathbf{x}, \mathbf{z}, \boldsymbol{\lambda}). \tag{5.9}$$

We first introduce some additional notation. Let  $\gamma_i$  be the vector of log-likelihood ratios that correspond to the  $i$ -th symbol of the code. In other words,

$$\gamma_i = (\gamma_{(i-1)(q-1)+1}, \gamma_{(i-1)(q-1)+2}, \dots, \gamma_{i(q-1)}).$$

Let  $\lambda_j^{(i)}$  be the length- $(q-1)$  sub-vector of  $\lambda_j$  that corresponds to the  $i$ -th symbol of the code. Similarly, we define  $z_j^{(i)}$  to be the sub-vector of  $z_j$  that correspond to the  $i$ -th symbol of the code. We then write  $\mathcal{L}_\mu(\mathbf{x}, \mathbf{z}, \boldsymbol{\lambda})$  as

$$\mathcal{L}_\mu(\mathbf{x}, \mathbf{z}, \boldsymbol{\lambda}) = \sum_{i=1}^N \left( \gamma_i^T \mathbf{x}_i + \sum_{j \in \mathcal{N}_v(i)} \lambda_j^{(i)T} (\mathbf{x}_i - \mathbf{z}_j^{(i)}) + \frac{\mu}{2} \sum_{j \in \mathcal{N}_v(i)} \|\mathbf{x}_i - \mathbf{z}_j^{(i)}\|_2^2 \right).$$

Note that when  $\mathbf{z}_j$  and  $\lambda_j$  are fixed for all  $j \in \mathcal{J}$ , we can decouple  $\mathbf{x}_i$  for all  $i \in \mathcal{I}$  in the sense that they can be individually solved for. Therefore

$$\begin{aligned} \mathbf{x}_i^{k+1} &= \operatorname{argmin}_{\mathbf{x}_i \in \mathbb{S}_{q-1}} \gamma_i^T \mathbf{x}_i + \sum_{j \in \mathcal{N}_v(i)} \lambda_j^{(i)T} (\mathbf{x}_i - \mathbf{z}_j^{(i)}) + \frac{\mu}{2} \sum_{j \in \mathcal{N}_v(i)} \|\mathbf{x}_i - \mathbf{z}_j^{(i)}\|_2^2 \\ &= \operatorname{argmin}_{\mathbf{x}_i \in \mathbb{S}_{q-1}} \|\mathbf{x}_i - \mathbf{v}_i\|_2^2, \end{aligned} \quad (5.10)$$

where

$$\mathbf{v}_i = \frac{1}{|\mathcal{N}_v(i)|} \left[ \sum_{j \in \mathcal{N}_v(i)} \left( \mathbf{z}_j^{(i)} - \frac{\lambda_j^{(i)}}{\mu} \right) - \frac{\gamma_i}{\mu} \right]. \quad (5.11)$$

Thus, the  $\mathbf{x}$ -update is equivalent to a Euclidean projection onto  $\mathbb{S}_{q-1}$ , which is solvable in linear time (i.e.,  $O(q)$ ) using techniques proposed in [23]. Note that the  $\mathbf{v}_i$  in (5.11) is an average of the corresponding replicas (i.e.,  $\mathbf{z}_j^{(i)}$ ) plus adjustments from the Lagrange multipliers (i.e.,  $\lambda_j^{(i)}$ ) and the log-likelihood ratios (i.e.,  $\gamma_i$ ).

In the  $\mathbf{z}$ -update we can solve for each  $\mathbf{z}_j$  separately. When  $\mathbf{x}_i$  and  $\lambda_j$  are fixed for all  $i \in \mathcal{I}$  and  $j \in \mathcal{J}$  we complete the square with respect to each  $\mathbf{z}_j$  and obtain the following update rule:

$$\mathbf{z}_j^{k+1} = \operatorname{argmin}_{\mathbf{z}_j \in \mathbb{V}_j} \|\mathbf{u}_j - \mathbf{z}_j\|_2^2, \quad (5.12)$$

where  $\mathbf{u}_j = \mathbf{P}_j \mathbf{x} + \lambda_j / \mu$ . Thus the  $\mathbf{z}$ -update is equivalent to the Euclidean projection onto  $\mathbb{V}_j$ .

For **LP-FR**, the respective  $\mathbf{z}$ -update is equivalent to projection onto  $\mathbb{U}_j$ . That is,

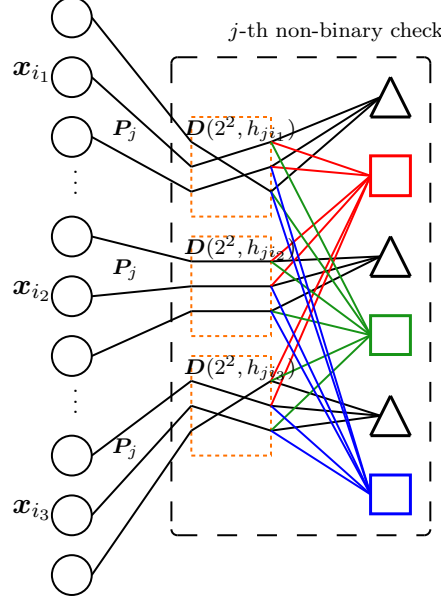
$$\mathbf{z}_j^{k+1} = \operatorname{argmin}_{\mathbf{z}_j \in \mathbb{U}_j} \|\mathbf{u}_j - \mathbf{z}_j\|_2^2, \quad (5.13)$$

In our previous work [25], we proposed an ADMM algorithm to solve (5.13). We show in Section 5.4 that the complexity of the decoding algorithm in [25] can be greatly reduced.

## 5.4 Improved ADMM LP decoding for LP-FR

In this section, we focus on **LP-FR** and propose an ADMM decoding algorithm that does not require a sub-routine that projects onto  $\mathbb{U}$ . We note that it is easy to extend the techniques in this section to problem **LP-CR**.

For an LDPC code, all entries of the embedded vector of a non-binary variable  $i \in \mathcal{I}$  participate in  $|\mathcal{N}_v(i)|$  non-binary checks. When embedding is used, each non-binary check can be decomposed



**Figure 2:** Factor graph of embeddings showing two hierarchies. The code under consideration is in  $\mathbb{F}_{2^2}$ .

by at-most-one-on-checks and parity-checks. We may think of these constraints having two hierarchies. The first hierarchy consists of constraints defined by the parity-check matrix of a non-binary code. The second hierarchy consists of constraints defined by the SPC code embeddings (i.e., Definition 15). We illustrated this in Example 30.

**Example 30** We build the factor graph of embeddings for an LDPC code in  $\mathbb{F}_{2^2}$ . Assume that the  $j$ -th check is connected to three variable nodes  $i_1$ ,  $i_2$  and  $i_3$ . Then, the embedded variables (9 binary variables) are connected to the  $j$ -th non-binary check. This connection is captured by the matrix  $P_j$  in (5.1). We can think of this as the first hierarchy. There is another hierarchy of factor graph inside the  $j$ -th constraint set, namely, those constraints defined by Lemma 12. The variables are first permuted by the rotation steps defined in Section 4.3 where the permutations are captured by the matrix  $D$ . Then, they are connected to a normalized constraint set defined by three at-most-one-on-checks and three parity-checks. The three at-most-one-on-checks are connected to the vector triplets  $\mathbf{x}_{i_1}$ ,  $\mathbf{x}_{i_2}$  and  $\mathbf{x}_{i_3}$  respectively. The three parity-check are specified by  $\tilde{B}(\mathcal{K}, 1)$  for  $\mathcal{K} = \{1\}, \{2\}$  and  $\{1, 2\}$ ; and are connected to the **permuted** entries of  $\mathbf{x}_{i_1}$ ,  $\mathbf{x}_{i_2}$  and  $\mathbf{x}_{i_3}$ . These connections are shown in Figure 2.

We note that we obtain Figure 2 by going through two hierarchies of constraints. However, the resulting graph is nothing but a factor graph of embeddings with two types of factor nodes: at-most-one-on-check nodes and parity-check nodes. The connections between variable nodes and factor nodes are specified by both the parity-check matrix of the code and the constraints specified for SPC codes (i.e., those in Lemma 12).

We now formally state the ADMM formulation based on the factor graph of embeddings. We will assume that the code has regular variable degree  $d_v$  for conciseness. However, our derivation can easily extend to irregular codes. By Lemma 21, we can rewrite the LP decoding problem

using the normalized polytope  $\mathbb{U}^N$  and rotation matrix  $\mathbf{D}$ . From now on, we use  $\mathbf{D}_j$  to denote the rotation matrix  $\mathbf{D}(q, \mathbf{h}_j)$ , where  $\mathbf{h}_j$  is the non-zero sub-vector of the  $j$ -th check of the code. Then, **LP-FR** can be rewritten as

$$\min \gamma^T \mathbf{x} \quad \text{subject to } \mathbf{D}_j^{-1} \mathbf{P}_j \mathbf{x} \in \mathbb{U}^N, \text{ for all } j \in \mathcal{J}, \quad (5.14)$$

Note that  $\mathbb{U}^N$  is defined as the intersection of simplexes (condition (a) and (b) in Definition 15) and parity polytopes (condition (c) in Definition 15). For condition (c) we define a select-and-add matrix  $\mathbf{T}_k$  using the following procedure. We first let  $\mathcal{K}_k$ ,  $k \in [2^m - 1]$ , be all non-empty subsets of  $[m]$ . We then use  $\mathbf{T}_k$  to denote the matrix that selects the entries in  $\tilde{\mathcal{B}}(\mathcal{K}_k, 1)$  and adds them to form the vector  $\mathbf{g}^{\mathcal{K}_k}$ . As a result, we can rewrite (5.14) as

$$\begin{aligned} \min \quad & \gamma^T \mathbf{x} \\ \text{subject to} \quad & \mathbf{T}_k \mathbf{D}_j^{-1} \mathbf{P}_j \mathbf{x} \in \mathbb{P}_{d_c}, \text{ for all } j \in \mathcal{J} \text{ and } k \in [2^m - 1] \\ & \mathbf{x}_i \in \mathbb{S}_{q-1} \text{ for all } i \in \mathcal{I}. \end{aligned} \quad (5.15)$$

We introduce replicas  $\mathbf{z}_{j,k}$  and  $\mathbf{s}_i$ . We let  $\mathbf{z}_{j,k} = \mathbf{T}_k \mathbf{D}_j^{-1} \mathbf{P}_j \mathbf{x}$  and  $\mathbf{s}_i = \mathbf{x}_i$ . However, note that we draw a distinction between  $\mathbf{z}$  and  $\mathbf{s}$  only for notation purposes. Both  $\mathbf{z}$  and  $\mathbf{s}$  serve the same purpose (i.e., replicas) in the ADMM algorithm.

$$\begin{aligned} \min \quad & \gamma^T \mathbf{x} \\ \text{subject to} \quad & \text{for all } j \in \mathcal{J}, k \in [2^m - 1] \text{ and } i \in \mathcal{I}, \\ & \mathbf{z}_{j,k} = \mathbf{T}_k \mathbf{D}_j^{-1} \mathbf{P}_j \mathbf{x}, \\ & \mathbf{z}_{j,k} \in \mathbb{P}_{d_c}, \\ & \mathbf{s}_i = \mathbf{x}_i, \\ & \mathbf{s}_i \in \mathbb{S}_{q-1}. \end{aligned} \quad (5.16)$$

By defining  $\mathbf{Z}_{j,k} := \mathbf{T}_k \mathbf{D}_j^{-1} \mathbf{P}_j$ , we can write the augmented Lagrangian as

$$\begin{aligned} \mathcal{L}_\mu(\mathbf{x}, \mathbf{z}, \mathbf{s}, \boldsymbol{\lambda}, \boldsymbol{\eta}) = & \gamma^T \mathbf{x} + \sum_{j,k} \boldsymbol{\lambda}_{j,k}^T (\mathbf{Z}_{j,k} \mathbf{x} - \mathbf{z}_{j,k}) + \frac{\mu}{2} \sum_{j,k} \|\mathbf{Z}_{j,k} \mathbf{x} - \mathbf{z}_{j,k}\|_2^2 \\ & + \sum_i \boldsymbol{\eta}_i^T (\mathbf{S}_i \mathbf{x} - \mathbf{s}_i) + \frac{\mu}{2} \sum_i \|\mathbf{S}_i \mathbf{x} - \mathbf{s}_i\|_2^2, \end{aligned} \quad (5.17)$$

where  $\mathbf{S}_i$  selects the sub-vector  $\mathbf{x}_i$  from  $\mathbf{x}$ . The ADMM updates for this problem are

$$\begin{aligned} \mathbf{x}\text{-update: } \mathbf{x}^* &= \operatorname{argmin}_{\mathbf{x}} \mathcal{L}_\mu(\mathbf{x}, \mathbf{z}, \mathbf{s}, \boldsymbol{\lambda}, \boldsymbol{\eta}), \\ \mathbf{z}\text{-update: } \mathbf{z}^* &= \operatorname{argmin}_{\mathbf{z} \in \mathbb{P}_{d_c}} \mathcal{L}_\mu(\mathbf{x}^*, \mathbf{z}, \mathbf{s}, \boldsymbol{\lambda}, \boldsymbol{\eta}), \\ \mathbf{s}\text{-update: } \mathbf{s}^* &= \operatorname{argmin}_{\mathbf{s} \in \mathbb{S}_{q-1}} \mathcal{L}_\mu(\mathbf{x}^*, \mathbf{z}^*, \mathbf{s}, \boldsymbol{\lambda}, \boldsymbol{\eta}), \\ \boldsymbol{\lambda}\text{-update: } \boldsymbol{\lambda}_j^* &= \boldsymbol{\lambda}_j^k + \mu (\mathbf{Z}_{j,k} \mathbf{x}^* - \mathbf{z}_{j,k}^*), \\ \boldsymbol{\eta}\text{-update: } \boldsymbol{\eta}_j^* &= \boldsymbol{\eta}_j^k + \mu (\mathbf{S}_i \mathbf{x}^* - \mathbf{s}_i^*). \end{aligned}$$

We make the following remarks. First, the  $\mathbf{x}$ -update is an unconstrained optimization problem, which is different from the case in Section 5.3. Second, we separate the  $\mathbf{z}$ -update and the  $\mathbf{s}$ -update because the two types of replicas are in two different polytopes ( $\mathbb{P}_{d_c}$  and  $\mathbb{S}_{q-1}$  respectively).

### 5.4.1 $\mathbf{x}$ -update

By taking the gradient of  $\mathcal{L}_\mu(\mathbf{x}, \mathbf{z}, \mathbf{s}, \boldsymbol{\lambda})$  and setting it to the all-zeros vector we obtain the following  $\mathbf{x}$ -update

$$\mathbf{x}^* = (\mathbf{Z} + \mathbf{I})^{-1} \left( \sum_{j,k} \mathbf{Z}_{j,k}^T (\mathbf{z}_{j,k} - \frac{\boldsymbol{\lambda}_{j,k}}{\mu}) + \sum_i \mathbf{S}_i^T (\mathbf{s}_i - \frac{\boldsymbol{\eta}_i}{\mu}) - \frac{\boldsymbol{\gamma}}{\mu} \right),$$

where  $\mathbf{Z} = \sum_{j,k} \mathbf{Z}_{j,k}^T \mathbf{Z}_{j,k}$  and the identity matrix  $\mathbf{I}$  comes from the fact that  $\sum_i \mathbf{S}_i^T \mathbf{S}_i = \mathbf{I}$ . Note that  $\mathbf{T}_k^T$  has at most one 1 in each row due to constraints defined in Definition 15. Since the permutation matrix  $\mathbf{D}_j^{-1}$  and the selection matrix  $\mathbf{P}_j$  all have at most one 1 in each column,  $\mathbf{Z}_{j,k}^T$  simply selects the entries in  $\mathbf{z}_{j,k}$  that correspond to some entries in  $\mathbf{x}$ <sup>7</sup>. We use the notation  $\mathbf{z}_{j,k}^{(i)}$  to represent the length- $(q-1)$  sub-vector of  $\mathbf{z}_{j,k}$  that corresponds to  $\mathbf{x}_i$ . Therefore, we can rewrite  $\mathbf{x}$ -update as

$$\mathbf{x}^* = (\mathbf{Z} + \mathbf{I})^{-1} \mathbf{t},$$

where  $\mathbf{t} = (\mathbf{t}_1, \dots, \mathbf{t}_N)$  and

$$\mathbf{t}_i = \sum_{(j,k) \in \mathcal{N}_v(i)} \left( \mathbf{z}_{j,k}^{(i)} - \frac{\boldsymbol{\lambda}_{j,k}^{(i)}}{\mu} \right) + \mathbf{s}_i - \frac{\boldsymbol{\eta}_i}{\mu} - \frac{\boldsymbol{\gamma}_i}{\mu}. \quad (5.18)$$

Note that the matrix  $(\mathbf{Z} + \mathbf{I})^{-1}$  is fixed by the code. Therefore one needs to calculate it only once per code. However, a naive calculation of  $(\mathbf{Z} + \mathbf{I})^{-1} \mathbf{t}$  has complexity  $O(N^2(2^m - 1)^2)$  which can be prohibitive for large  $N$ . We therefore introduce a method to calculate this product in linear complexity  $O(N(2^m - 1))$ .

**Lemma 31**  $(\mathbf{Z} + \mathbf{I})^{-1} \mathbf{t}$  can be calculated in linear complexity  $O(N(2^m - 1))$ .

**Proof** See Appendix 11. ■

### 5.4.2 Other ADMM updates

The  $\mathbf{z}$ - and  $\mathbf{s}$ -updates are the same as (5.12) except that we now perform the respective projections onto  $\mathbb{P}_{d_c}$  and  $\mathbb{S}_{2^m-1}$ .

$$\begin{aligned} \mathbf{s}_i^* &= \operatorname{argmin}_{\mathbf{s}_i \in \mathbb{S}_{2^m-1}} \|\mathbf{u}_i - \mathbf{s}_i\|_2^2 \text{ for } i \in \mathcal{I} \\ \mathbf{z}_{j,k}^* &= \operatorname{argmin}_{\mathbf{z}_{j,k} \in \mathbb{P}_{d_c}} \|\mathbf{v}_{j,k} - \mathbf{z}_{j,k}\|_2^2 \text{ for } j \in \mathcal{J} \text{ and } k \in [2^m - 1], \end{aligned}$$

where  $\mathbf{u}_i = \mathbf{S}_i \mathbf{x} + \boldsymbol{\eta}_i/\mu$  and  $\mathbf{v}_{j,k} = \mathbf{Z}_{j,k} \mathbf{x} + \boldsymbol{\lambda}_{j,k}/\mu$ . Both projection operations have complexity that is linear in  $q(=2^m)$  and  $d_c$  respectively (cf. [14] and [23]).

**Proposition 32** Let  $\tilde{\mathbf{x}}$  be the solution of the LP decoding problem (5.2). For any  $\eta > 0$ , Algorithm 1 will, in  $O(Nq^2)$  time, determine a vector  $\hat{\mathbf{x}}$  that satisfies the constraints in (5.2) and that also satisfies the following bound:

$$\mathbf{L}_v(\mathbf{y})^T \hat{\mathbf{x}} - \mathbf{L}_v(\mathbf{y})^T \tilde{\mathbf{x}} < qNd_c\eta,$$

where  $\mathbf{y}$  is the channel output,  $q = 2^m$  is the field size and  $d_c$  is the check degree.

---

<sup>7</sup>Recall that  $\mathbf{Z}_{j,k}$  maps  $\mathbf{x}$  to the replica  $\mathbf{z}_{j,k}$ . In addition, there is at most one 1 per column in  $\mathbf{Z}_{j,k}$ . This implies that  $\mathbf{Z}_{j,k}^T$  has at most one 1 per row. Thus,  $\mathbf{Z}_{j,k}^T$  simply selects entries from  $\mathbf{z}_{j,k}$ .

**Proof** This proposition is similar to Proposition 1 in [12]. We omit the details but point out that ADMM takes  $O(1)$  iterations to converge to a point with  $qNM\eta$  gap to optimal. For each iteration, the complexity of the  $\mathbf{x}$ -update is  $O(Nq^2)$ . The complexity of each  $(\mathbf{z}, \mathbf{s})$ -update is  $O(Nq + M(q-1)d_c)$ . Since  $d_c$  does not scale, we deduce that the complexity of each iteration is  $O(Nq^2)$ . Therefore due to the  $O(1)$  requirement on the number of iterations, the overall complexity is  $O(Nq^2)$ .  $\blacksquare$

We summarize ADMM LP decoding in Algorithm 1.

---

**Algorithm 1** ADMM LP decoding of LDPC codes in  $\mathbb{F}_{2^m}$

---

```

1: Construct the  $d_c \times (2^m - 1)N$  matrices  $\mathbf{Z}_{j,k}$  for all  $j \in \mathcal{J}$  and  $k \in [2^m - 1]$  based on the factor
   graph of embeddings.
2: Construct the  $(2^m - 1) \times (2^m - 1)N$  matrices  $\mathbf{S}_i$  for all  $i \in \mathcal{I}$  based on the at-most-one-on
   constraints.
3: Initialize all entries of  $\lambda_{j,k}$  and  $\eta_i$  to 0. Initialize all entries of  $\mathbf{z}_{j,k}$  and  $\mathbf{s}_i$  to  $\frac{1}{2^m}$ . Initialize
   iterate  $\delta = 0$ . To simplify the notation, we drop iterate  $\delta$  except when determining the stopping
   criteria.
4: repeat
5:   for all  $i = 1, \dots, N$  do
6:     Compute vector  $\mathbf{t}_i$  using equation (5.18). Store  $\|\mathbf{t}_i\|_1$ .
7:     Compute the variables  $a$  and  $b$  per Lemma 59.
8:     Update  $\mathbf{x}_i \leftarrow (a - b)\mathbf{t}_i + b\|\mathbf{t}_i\|_1$ .
9:   end for
10:  for all  $j = 1, \dots, M$  and  $k = 1, \dots, 2^m - 1$  do
11:    Set  $\mathbf{v}_{j,k} = \mathbf{Z}_{j,k}\mathbf{x} + \lambda_{j,k}/\mu$ .
12:    Update  $\mathbf{z}_{j,k} \leftarrow \Pi_{\mathbb{P}_{d_c}}(\mathbf{v}_{j,k})$  where  $\Pi_{\mathbb{P}_{d_c}}(\cdot)$  is a projection onto the parity polytope of dimension  $d_c$ .
13:    Update  $\lambda_{j,k} \leftarrow \lambda_{j,k} + \mu(\mathbf{Z}_{j,k}\mathbf{x} - \mathbf{z}_{j,k})$ .
14:  end for
15:  for all  $i = 1, \dots, N$  do
16:    Set  $\mathbf{u}_i = \mathbf{S}_i\mathbf{x} + \eta_i/\mu$ .
17:    Update  $\mathbf{s}_i \leftarrow \Pi_{\mathbb{S}_{q-1}}(\mathbf{u}_i)$  where  $\Pi_{\mathbb{S}_{q-1}}(\cdot)$  is a projection onto  $\mathbb{S}_{q-1}$ .
18:    Update  $\eta_i \leftarrow \eta_i + \mu(\mathbf{S}_i\mathbf{x} - \mathbf{s}_i)$ .
19:  end for
20:   $\delta \leftarrow \delta + 1$ .
21: until  $\sum_i \|\mathbf{S}_i\mathbf{x}^\delta - \mathbf{s}_i^\delta\|_2^2 + \sum_{j,k} \|\mathbf{Z}_{j,k}\mathbf{x}^\delta - \mathbf{z}_{j,k}^\delta\|_2^2 < \epsilon^2((2^m - 1)N + M(2^m - 1)d_c)$ 
   and  $\sum_i \|\mathbf{s}_i^\delta - \mathbf{s}_i^{\delta-1}\|_2^2 + \sum_{j,k} \|\mathbf{z}_{j,k}^\delta - \mathbf{z}_{j,k}^{\delta-1}\|_2^2 < \epsilon^2((2^m - 1)N + M(2^m - 1)d_c)$ 
   return  $\mathbf{x}$ .
```

---

### 5.4.3 Early termination and over-relaxation

In practice, ADMM can be terminated whenever a codeword is found. Doing so often reduces the number of iterations needed to decode. We observe empirically that early termination does affect the WER for some very short block length codes. However, it does not create observable effects for the codes we study in Section 7. We use the vectors  $\mathbf{s}_i$  for all  $i \in \mathcal{I}$  to determine whether or not

the corresponding non-binary vector is a codeword. The decision process is as follows. For each  $i \in \mathcal{I}$ , we first obtain the vector  $\hat{\mathbf{x}} := (\hat{x}_0, \hat{x}_1, \dots, \hat{x}_{2^m-1})$  by letting  $\hat{x}_0 = 1 - \|\mathbf{s}_i\|$  and  $\hat{x}_k = s_{i,k}$  for all  $k \in [2^m - 1]$ . Then, the  $i$ -th non-binary symbol is determined by  $\hat{c}_i = \arg\max_k \hat{x}_k$ . After we have obtained the length- $N$  non-binary vector  $\hat{\mathbf{c}}$ , we terminate ADMM if  $\mathbf{H}\hat{\mathbf{c}} = 0$  in  $\mathbb{F}_{2^m}$  where  $\mathbf{H}$  is the parity-check matrix of the code.

Another useful technique in practice is over-relaxation, which is already used in [12]. Readers are referred to [26] and references therein for details on over-relaxation. We observe that over-relaxation is effective for Algorithm 1. Experiments surrounding the choices of over-relaxation parameter  $\rho$  are presented in Section 7.

## 6 ADMM penalized decoding of non-binary codes

In this section, we introduce an ADMM penalized decoder for non-binary codes. This decoder is analogous to the penalized decoder for binary codes introduced in [15]. It is shown in [15] that adding a non-convex penalty term to the LP decoding objective can improve the low SNR performance of LP decoding. The penalty term penalizes fractional solutions and can improve the performance because correct solutions should be integral. Herein, we extend this idea to embeddings of non-binary symbols.

One important characteristic of linear codes is that, at least when codewords are transmitted over a symmetric channel, the probability of decoding failure should be independent of the codeword that is transmitted. To get this property to hold for the non-binary penalized decoder, herein, we use the constant-weight embedding. This desired “codeword-independent” property follows because all non-binary symbols are embedded symmetrically with respect to each other. As a reminder, for Flanagan’s embedding, symbol 0 is treated differently from others symbol in the field. We discuss this issue in details in Section 6.2.

### 6.1 ADMM penalized decoding algorithm

Formally, we consider the following decoding problem:

$$\begin{aligned} \min \quad & \boldsymbol{\gamma}'^T \mathbf{x} - \alpha \sum_i \|\mathbf{x}_i - \mathbf{r}\|_2^2 \\ \text{subject to} \quad & \mathbf{P}'_j \mathbf{x} \in \mathbb{U}'_j, \forall j \in \mathcal{J}, \end{aligned} \tag{6.1}$$

where  $\mathbb{U}'_j$  is the relaxed polytope under the constant-weight embedding (Definition 38),  $\boldsymbol{\gamma}' := \mathbf{L}'_v(\mathbf{y})$  and

$$\mathbf{r} = \left( \frac{1}{q}, \frac{1}{q}, \dots, \frac{1}{q} \right).$$

We pick the  $\ell_2$  norm as the penalty in (6.1) because it yields simple ADMM update rules and because it yields good empirical performance (cf. Section 7). For both the  $\mathbf{x}$ -update and the  $\mathbf{z}$ -update, we complete the square and then perform minimizations. As a result, the  $\mathbf{x}$ -update can be expressed as a projection onto  $\mathbb{S}'_q$ ; and the  $\mathbf{z}$ -update can be expressed as a projection onto  $\mathbb{U}'_j$ . In particular, the  $\mathbf{x}$ -update rule is

$$\mathbf{x}_i^* = \Pi_{\mathbb{S}'_q} \left[ \frac{1}{d_i - \frac{2\alpha}{\mu}} \left( \sum_{j \in \mathcal{N}_v(i)} \left( \mathbf{z}_j^{(i)} - \frac{\boldsymbol{\lambda}_j^{(i)}}{\mu} \right) - \frac{\boldsymbol{\gamma}'_i}{\mu} - \frac{2\alpha \mathbf{r}}{\mu} \right) \right].$$

The  $\mathbf{z}$ -update can be written as a projection onto  $\mathbb{U}'$ . For this projection operation, we use the rotate-and-project technique introduced in Section 4.3 (see also [25] for more details). The ADMM penalized decoder is summarized in Algorithm 2.

---

**Algorithm 2** ADMM penalized decoding. **Input:** Received vector  $\mathbf{y} \in \Sigma^N$ . **Output:** Decoded vector  $\mathbf{x}$ .

---

- 1: Construct the  $qd_j \times qN$  selection matrix  $\mathbf{P}'_j$  for all  $j \in \mathcal{J}$  based on the parity-check matrix  $\mathbf{H}$ .
  - 2: Construct the log-likelihood ratio  $\mathbf{L}'_v(\mathbf{y})$ .
  - 3: For all  $j \in \mathcal{J}$ , initialize all entries of  $\boldsymbol{\lambda}_j$  to 0 and initialize all entries of  $\mathbf{z}_j$  to 0.5. Initialize iterate  $\delta = 0$ . To simplify the notation, we drop iterate  $\delta$  except when determining the stopping criteria.
  - 4: **repeat**
  - 5:   **for all**  $i \in \mathcal{I}$  **do**
  - 6:     Update  $\mathbf{x}_i$  by
 
$$\mathbf{x}_i = \Pi_{\mathbb{S}'_q} \left[ \frac{1}{d_i - \frac{2\alpha}{\mu}} \left( \sum_{j \in \mathcal{N}_v(i)} \left( \mathbf{z}_j^{(i)} - \frac{\boldsymbol{\lambda}_j^{(i)}}{\mu} \right) - \frac{\boldsymbol{\gamma}'_i}{\mu} - \frac{2\alpha\mathbf{r}}{\mu} \right) \right]$$
  - 7:   **end for**
  - 8:   **for all**  $j \in \mathcal{J}$  **do**
  - 9:     Set  $\mathbf{v}_j \leftarrow \mathbf{P}'_j \mathbf{x} + \boldsymbol{\lambda}_j / \mu$ .
  - 10:    Update  $\mathbf{z}_j \leftarrow \Pi_{\mathbb{U}'_j}(\mathbf{v}_j)$  where  $\Pi_{\mathbb{U}'_j}(\cdot)$  is a projection onto the relaxed code polytope defined by the  $j$ -th check.
  - 11:    Update  $\boldsymbol{\lambda}_j \leftarrow \boldsymbol{\lambda}_j + \mu (\mathbf{P}'_j \mathbf{x} - \mathbf{z}_j)$ .
  - 12:   **end for**
  - 13:    $\delta \leftarrow \delta + 1$ .
  - 14: **until**  $\sum_j \|\mathbf{P}'_j \mathbf{x}^\delta - \mathbf{z}_j^\delta\|_2^2 < \epsilon^2 q M d_j$   
     and  $\sum_j \|\mathbf{z}_j^\delta - \mathbf{z}_j^{\delta-1}\|_2^2 < \epsilon^2 q M d_j$   
   **return**  $\mathbf{x}$ .
- 

**Theorem 33** *Under the channel symmetry condition in the sense of [16], the probability that Algorithm 2 fails is independent of the codeword that was transmitted.*

**Proof** See Appendix 10.10 ■

## 6.2 Discussions

The penalized decoder described above requires a sub-routine that projects onto the code polytope. The ADMM projection technique introduced in [25] scales linearly with  $dq^2$ , where  $d$  is the degree of the check node and  $q$  is the field size. However, this projection technique is iterative and inaccurate, because it accepts an error tolerance  $\epsilon_p$ . As a result, the provable number of iterations scales linearly with  $\frac{1}{\epsilon_p}$  due to the linear convergence rate of ADMM (cf. [27]). This means that each iteration of the ADMM penalized decoder scales linearly with  $1/\epsilon_p$ . On the contrary, each iteration of Algorithm 1 does not depend on such error tolerance. As a result, the computational complexity of the penalized decoder is much higher than the ADMM LP decoder in Algorithm 1. We observe



empirically that in our implementation of ADMM penalized decoding is around 20 times slower than the ADMM LP decoder in Algorithm 1<sup>8</sup>.

We note that Algorithm 1, in fact, can be used to try to solve the penalized objective (6.1). However, we observe empirically that the resulting decoder does not have the codeword symmetry property in the sense of Theorem 33. We briefly discuss why the proof technique of Theorem 33 cannot be applied to this decoder. First, note that the penalized decoder tries to solve a non-convex program. As a result, the output of the penalized decoder cannot be determined solely by the optimization problem. It depends on the ADMM update rules (the  $\mathbf{x}$ -,  $\mathbf{z}$ - and  $\boldsymbol{\lambda}$ -update), which are determined by the way the optimization problem is stated<sup>9</sup>. In the proof of Theorem 33, we show that the decoding process when decoding the all-zeros codeword is “symmetric” with respect to any other codeword’s decoding process in the sense of Lemma 54 (see rigorous discussions in Appendix 10.10). This symmetry behavior is due to the symmetric structure of  $\mathbb{U}'$  (Lemma 52). If we were to use Algorithm 1 to try to solve (6.1), the polytopes used to describe the constraint set are parity polytopes and simplexes, not  $\mathbb{U}'$ . As a result, we cannot have the symmetry behavior for parity polytopes and simplexes in the sense of Lemma 52.

On the other hand, we note that applying Algorithm 1 to (6.1) results in a significant improvement in terms of decoding efficiency when compared to Algorithm 2. Furthermore, we observe empirically that this technique can achieve a much reduced error rate when compared to LP decoding (**LP-FR** and **LP-CR**) for the all-zeros codeword. Open questions include whether all codewords experience some improvement and how much improvement each codeword receives. These questions are non-trivial because of the non-convexity of the penalized objective.

## 7 Numerical results and discussions

In this section we present numerical results for ADMM LP decoding (Algorithm 1) and ADMM penalized LP decoding (Algorithm 2). First, we show the error rate performance of the two decoders as a function of SNR. In addition, we compare our decoders to the low-complexity LP (LCLP) decoding technique proposed by Punekar *et al.* in [19]. Next, in Section 7.2, we focus on choosing a good set of parameters for Algorithm 1. Finally, we show how the penalty coefficient  $\alpha$  (cf. (6.1)) affects the WER performance of ADMM penalized decoding.

### 7.1 Performance of the proposed decoders

In this subsection we simulate three codes and demonstrate their error rate performance as a function of SNR. The first code we simulate is derived from a binary length-2048 progressive edge growth (PEG) code that can be obtained from [28]. The other two codes are derived from two binary Tanner codes obtained from [29]. We select these codes for the following reasons. First, we select codes that are of different block lengths (2048, 1055 and 755) and in different fields ( $\mathbb{F}_{22}$  and  $\mathbb{F}_{23}$ ). Second, the two Tanner codes have been studied in [19]. Therefore we can make direct comparisons with the results therein. Last but not least, the parity-check matrices of these codes are easy to obtain (e.g., from [28]). Our intent is to make our simulations using these codes easy to repeat and thus to compare to.

---

<sup>8</sup>As measured in execution time (sec).

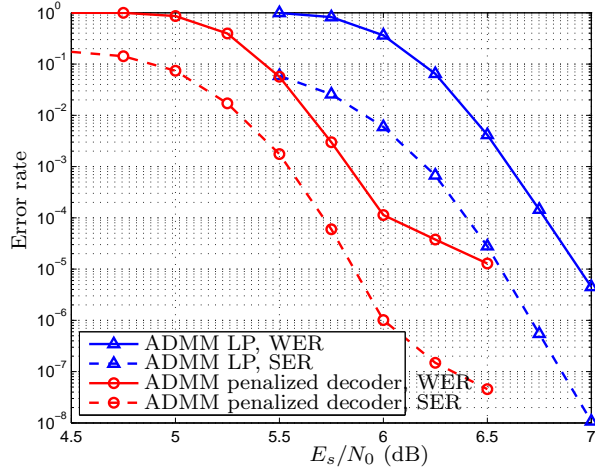
<sup>9</sup>That is, (6.1) can be rewritten into other equivalent forms. Each form may result in a different ADMM algorithm.

Figure 3 plots the word-error-rate (WER) and symbol-error-rate (SER) of a length-2048 LDPC code when decoded using ADMM LP decoding and ADMM penalized decoding. This code is derived from the PEG [2048, 1018] binary LDPC code obtained from [28]. We use the same parity-check matrix as the original PEG code except that we let each non-zero check value be  $1 (= \xi^0) \in \mathbb{F}_{2^2}$ . The code symbols are modulated using quaternary phase-shift keying (QPSK) and transmitted over an AWGN channel. Denote by  $(x, y)$  the in-phase and quadrature components. We modulate the symbols in the following way:  $0 \mapsto (1, 0)$ ,  $\xi^0 \mapsto (0, 1)$ ,  $\xi^1 \mapsto (-1, 0)$  and  $\xi^2 \mapsto (0, -1)$ . We use energy per information symbol ( $E_s/N_0$ ) as our unit of SNR. We show explicitly how we calculate  $E_s/N_0$  in Appendix 13. In this simulation, we decode using ADMM LP decoding (Algorithm 1) and ADMM penalized decoding (Algorithm 2). We note that both algorithms require many parameters. We show how to choose parameters in Sections 7.2 and 7.3. For ADMM LP decoding we use the following parameter settings: the “step size” of ADMM  $\mu = 2$  (cf. (5.17)); the maximum number of iterations  $T_{\max} = 200$ ; the ending tolerance  $\epsilon = 10^{-5}$  (cf. Algorithm 1), and the over-relaxation parameter  $\rho = 1.9$  (cf. Section 5.4.3). For ADMM penalized decoding, we let  $\mu = 4$ ,  $T_{\max} = 200$ ,  $\rho = 1.5$ ,  $\epsilon = 10^{-5}$ , and  $\alpha = 0.8$  (cf. (6.1)). For each data point, we collect more than 100 word errors.

We make the following observations. First, both decoding algorithms display a “waterfall” behavior in terms of error rates. However, penalized decoding initiates the waterfall at a much lower SNR (about 0.7dB lower in this example). This shows that ADMM penalized decoding significantly improves LP decoding at low SNRs. Second, ADMM penalized decoding displays an error-floor at WER  $10^{-4}$  (SER  $10^{-6}$ ). However, LP decoding does not display an error-floor for WER above  $10^{-5}$  (SER above  $10^{-8}$ ). Both observations are consistent with binary LP decoding and binary penalized decoding [15]. We note that unlike the case with binary decoders where the binary ADMM penalized decoder outperforms the binary ADMM LP decoding in terms of both the number of iterations and execution time, the average decoding time of Algorithm 2 using our implementation is much longer than that of Algorithm 1 due to the reasons discussed in Section 6. Although Algorithm 2 is competitive in terms of WER performance, it is not competitive in terms of execution time.

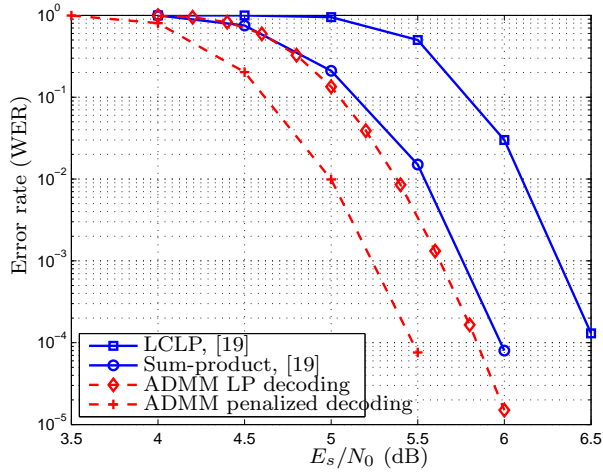
The second code we simulate is the length-1055 code used in [19]. This code is derived from the Tanner [1055, 424] binary LDPC code introduced by Tanner in [29]. Similar to the previous case, each binary non-zero entry is replaced by the value  $1 \in \mathbb{F}_{2^2}$ . We keep all other simulation settings the same as the previous case, i.e., we use the same AWGN channel with QPSK modulation and the same parameter settings for Algorithm 1. For the penalized decoder, we change the parameter settings. We use the following settings:  $\mu = 4$ ,  $\rho = 1.5$ ,  $T_{\max} = 100$ ,  $\epsilon = 10^{-5}$ , and  $\alpha = 0.6$ . We first plot the WER performance of the code in Figure 4. The performance of LCLP and the sum-product algorithm (SPA) decoders are plotted for comparison. The data for these decoders is obtained from [19].

We make the following observations. First, we observe that ADMM LP decoding outperforms both SPA and LCLP in terms of WER. ADMM LP decoding has a 0.6dB SNR gain when compared to the LCLP algorithm of [19]. This result is interesting because ADMM LP decoding uses the relaxed code polytope  $\mathbb{U}$ . Thus, one might expect that in general it would perform worse than Flanagan’s LP decoding algorithm. However, we recall that we validated numerically that  $\mathbb{U} = \mathbb{V}$  for  $\mathbb{F}_{2^2}$  (cf. Appendix 12). Figure 4 suggests that LCLP may have a 0.6dB SNR loss due to the approximations made in that algorithm (see [19] for details). Second, the penalized decoder improves the WER performance by 0.4dB. This effect is consistent with Figure 3. Third, we note



**Figure 3:** ADMM LP decoding and ADMM penalized decoding error rates plotted as a function of SNR for the  $[2048, 1018]$  PEG code in  $\mathbb{F}_2$ .

that none of the decoders used in Figure 4 displays an error-floor.

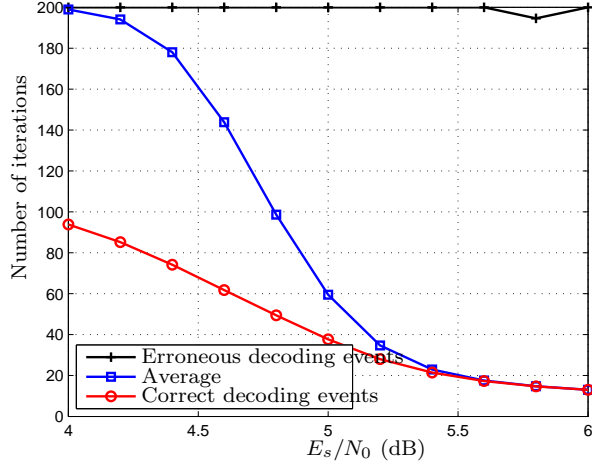


**Figure 4:** WER plotted as a function of SNR for the Tanner  $[1055, 424]$  code in  $\mathbb{F}_2$ .

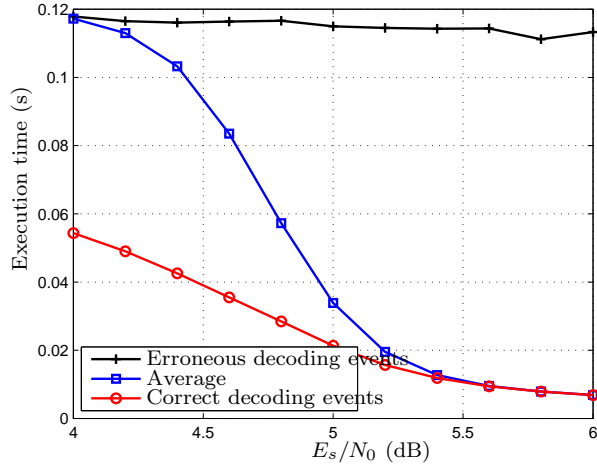
In Figure 5, we plot the average number of iterations for erroneous decoding events, all decoding events, and correct decoding events. We observe that for correct decoding events, the average number of iterations is less than 100 at all SNRs. This indicates that we can lower  $T_{\max}$  without greatly impacting WER. In fact, we observe that there is less than a 0.05dB loss if we use  $T_{\max} = 100$  (data not shown).

In Figure 6, we plot execution time statistics for the decoding events. In these simulations, the decoder is implemented using C++ and data is collected on a 3.10GHz Intel(R) Core(TM) i5-2400 CPU. We make the following observations: First, the average decoding time for correctly

decoded events decreases as the SNR increases. However, the average decoding time for erroneous decodings does not vary significantly with SNR. Second, the average time per decoding decreases rapidly at low SNRs. This is due to the waterfall behavior of the error rate. Third, we note that ADMM LP decoding (Algorithm 1) is a much faster algorithm in terms of execution time than ADMM penalized decoding (Algorithm 2). As an example, the average decoding time for ADMM LP decoding at  $E_s/N_0 = 5\text{dB}$  is 0.033s. In contrast, the average decoding time for ADMM penalized decoding at  $E_s/N_0 = 5\text{dB}$  is 0.61s (data not shown), which is about 15-20 times slower than Algorithm 1.

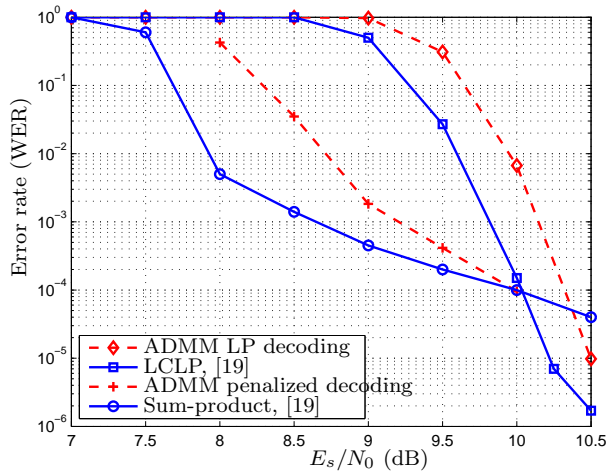


**Figure 5:** Number of iterations of Algorithm 1 plotted as a function of SNR for the Tanner [1055, 424] code in  $\mathbb{F}_{2^2}$ .



**Figure 6:** Software implementation execution time of Algorithm 1 plotted as a function of SNR for the Tanner [1055, 424] code in  $\mathbb{F}_{2^2}$ .

In Figure 7, we show results for the length-755 code also studied in [19]. This code has symbols in  $\mathbb{F}_{23}$  and is derived from the Tanner [755, 334] binary LDPC code [29]. We use the same non-binary parity-check matrix and the same 8-PSK modulation method as in [19]. We make the following remarks. First, ADMM LP decoding performs worse than LCLP in our simulations. This is due to the fact that ADMM LP decoding uses the relaxed code polytope. Unlike in the  $\mathbb{F}_{22}$  case, relaxing the polytope using Definition 15 induces a loss in terms of SNR performance. We observe that ADMM LP decoding is about 0.3dB worse than LCLP. Second, and similar to what we saw in the simulations for the previous two codes, the penalized decoder improves the low-SNR performance of LP decoding by around 1.5dB for  $E_s/N_0 \leq 9$ dB. However, the WER curve displays a significant error-floor. We observe that the optimal value for the penalty coefficient  $\alpha$  decreases as SNR increases (data not shown). At  $E_s/N_0 = 10$ dB, adding a positive penalty does not lead to an observable improvement in WER. Third, in comparison to Figure 4 in which no error-floor is observed for any decoder, we believe that the [755, 344] Tanner code is a problematic code. Thus, we think designing a good code for ADMM decoding is important future work.



**Figure 7:** WER plotted as a function of SNR for the Tanner [755, 344] code in  $\mathbb{F}_{23}$ .

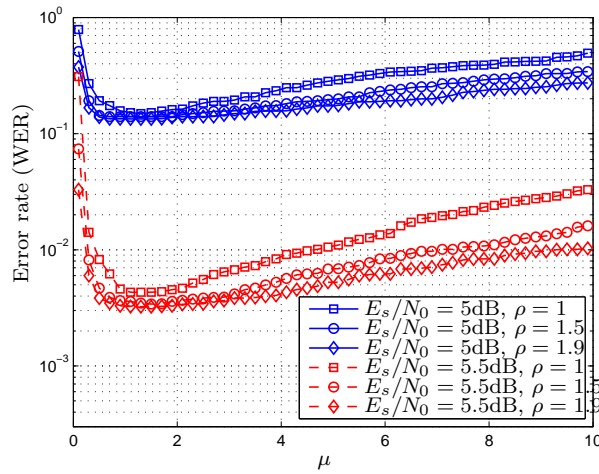
We make two concluding remarks. First, Algorithm 1 solves the relaxed LP decoding problem efficiently. It is competitive with earlier decoders in terms of WER performance, especially for  $\mathbb{F}_{22}$ . Second, the penalized decoder is a very promising decoder because it improves the low-SNR performance of LP decoding consistently. However, it suffers from high-complexity and may suffer from an early error-floor. It is important in future work to improve the complexity of the penalized decoder.

## 7.2 Parameter choices for ADMM LP decoding

In this section we study the effects of the choice of parameters for Algorithm 1. We conduct simulations based on the [1055, 424] Tanner code described in the previous subsection, and use the early termination technique described in Section 5.4.3. For each data point, we collect more than 100 word errors.

The ADMM algorithm has many parameters: the “step size” of ADMM  $\mu$  (cf. (5.17)), the maximum number of iterations  $T_{\max}$ , the ending tolerance  $\epsilon$  (cf. Algorithm 1), and the over-relaxation parameter  $\rho$  (cf. Section 5.4.3). We fix the number of iterations to be  $T_{\max} = 200$  and ending tolerance to be  $\epsilon = 10^{-5}$  because the decoder is less sensitive to the settings of these two parameters than it is to  $\mu$  or  $\rho$ .

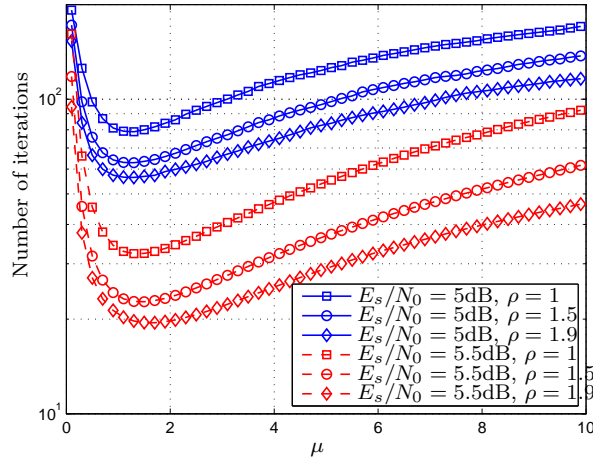
In Figures 8 and 9 we study the effects of the choice of parameter  $\mu$  on the decoder. In Figure 8 we plot the WER as a function of  $\mu$  when  $\rho = 1, 1.5$  and  $1.9$ . We ran simulations for  $E_s/N_0 = 5\text{dB}$  and  $E_s/N_0 = 5.5\text{dB}$ . In Figure 13 we plot the corresponding average number of iterations. We make the following observations. First, the WER does not change much as a function of  $\mu$  as long as  $\mu \in [1, 3]$ . In addition, the lowest number of iterations is attained when  $\mu \in [1, 3]$ . This means that a choice of  $\mu$  within the range  $[1, 3]$  is good for practical reasons. Second, the trend of the curves for different values of  $\rho$  is similar. In other words, the choice of  $\rho$  does not strongly affect the choice of  $\mu$ , at least empirically. This is useful because it simplifies the task of choosing good parameters. Third, we observe that the WER curves for both SNRs flatten when  $\mu \in [1, 3]$ <sup>10</sup>. This indicates the parameter choice for this data point allows ADMM to converge fast enough to achieve the LP decoding solution within the set number of iterations, at least in most cases. In particular, if we increase  $T_{\max}$ , the range of the flat region increases. We observe that for  $E_s/N_0 = 5\text{dB}$ ,  $T_{\max} = 1000$ , and  $\rho = 1$ , the WERs are between 0.122 and 0.128 for  $\mu \in [0.3, 3.1]$ . This also shows that in this case increasing  $T_{\max}$  does not greatly improve WER performance. Fourth, we observe that the WER performance of the decoder can be improved by assigning a large value to  $\rho$ . This effect is demonstrated further in Figures 10 and 11. Finally, we observe that the average numbers of iterations are small for both SNRs. Note that the SNRs in Figures 8 and 9 are low SNRs and the corresponding WERs are high. Recall that in Figure 5 the average number of iterations is much smaller than  $T_{\max} = 200$  for high SNRs. This means that  $T_{\max}$  does not need to be large.



**Figure 8:** ADMM step size  $\mu$ : WER plotted as a function of  $\mu$  for the Tanner [1055, 424] code in  $\mathbb{F}_{2^2}$ ,  $T_{\max} = 200$ , and  $\epsilon = 10^{-5}$ .

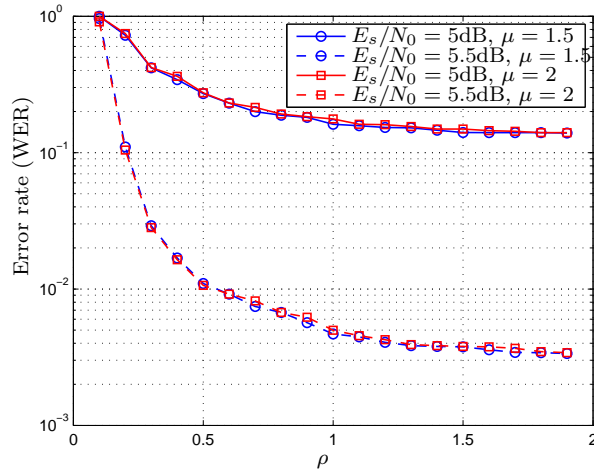
In Figures 10 and 11, we study the effect of the choice of parameter  $\rho$  on the decoder. In

<sup>10</sup>We use the same seed to generate noises for different values of  $\mu$ .



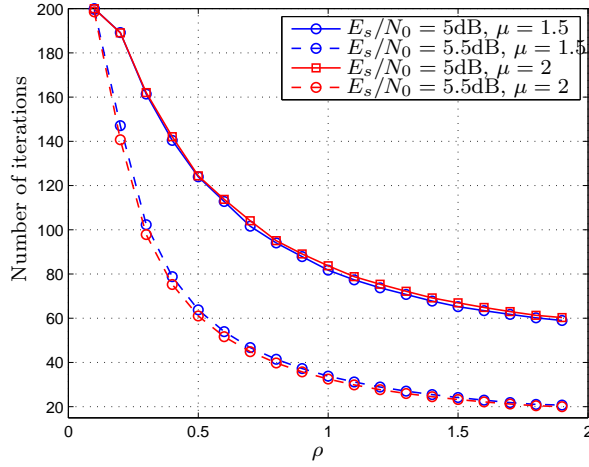
**Figure 9:** ADMM step size  $\mu$ : Number of iterations plotted as a function of  $\mu$  for the Tanner [1055, 424] code in  $\mathbb{F}_{2^2}$ ,  $T_{\max} = 200$ , and  $\epsilon = 10^{-5}$ .

Figure 10 we plot WER as a function of  $\rho$  when  $\mu$  takes on values 1.5 and 2. Again we simulate for  $E_s/N_0 = 5\text{dB}$  and  $E_s/N_0 = 5.5\text{dB}$ . In Figure 11, we plot the corresponding average number of iterations as a function of  $\rho$ . In these two figures we observe results that are consistent with Figures 8 and 9, i.e., the number of iterations decreases as  $\rho$  increases. However, the WER does not decrease significantly for  $\rho > 1$ .



**Figure 10:** Over-relaxation parameter  $\rho$ : WER plotted as a function of  $\rho$  for the Tanner [1055, 424] code in  $\mathbb{F}_{2^2}$ ,  $T_{\max} = 200$ , and  $\epsilon = 10^{-5}$ .

Finally, we compare Flanagan's embedding and the constant-weight embedding in terms of decoding performance. For Flanagan's embedding we use Algorithm 1. For the constant-weight embedding we apply the method in Algorithm 1 to the LP decoding problem (5.3). Due to the



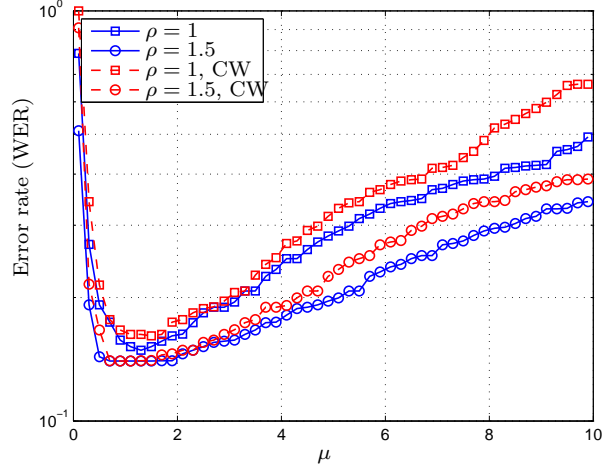
**Figure 11:** Over-relaxation parameter  $\rho$ : Number of iterations plotted as a function of  $\rho$  for the Tanner  $[1055, 424]$  code in  $\mathbb{F}_{2^2}$ ,  $T_{\max} = 200$ , and  $\epsilon = 10^{-5}$ .

similarity between Flanagan’s embedding and the constant-weight embedding, results in Section 5.4 can be easily extended to the constant-weight embedding. In Figures 12 and 13, we compare both WER and the average number of iterations for the two decoders. In this set of simulations we let  $E_s/N_0 = 5\text{dB}$ ,  $T_{\max} = 200$ , and  $\epsilon = 10^{-5}$ . In Figure 12 we observe that the WER for the constant-weight embedding is higher than it is for Flanagan’s embedding in most cases. The exception is the data points at  $\rho = 1.5$  and  $\mu \in [0.7, 1.7]$ , where both embeddings attain the same WERs. This exception is because that parameter choices allow ADMM to converge fast enough to obtain the LP decoding solution. Recall that we proved that the two embedding methods are equivalent in terms of LP decoding (cf. Corollary 28). It is not surprising that both decoders can achieve the same WERs when a sufficient number of iterations is allowed. From Figure 13 we observe that the number of iterations for the constant-weight embedding is greater than that required for Flanagan’s embedding. Recall that Flanagan’s embedding saves exactly 1 coordinate for each embedded vector when compared to the constant-weight embedding (cf. Section 3.1). As a result, the constant-weight embedding requires slightly more memory than Flanagan’s embedding. Thus, in practice, Flanagan’s embedding for LP decoding would be preferred.

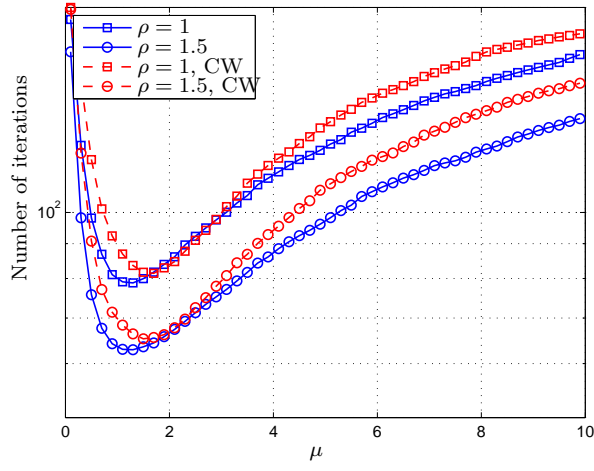
To summarize the above discussions, we emphasize four points:

- $\mu \in [1, 2]$  and  $\rho = 1.9$  are good parameter choices for the Tanner  $[1055, 424]$  code.
- The error performance of the decoder only weakly depends on parameter settings.
- Flanagan’s embedding is slightly better than the constant-weight embedding in terms of computational complexity and memory usage.
- The effects of the choice of parameters for the PEG  $[2048, 1018]$  code discussed in Section 7.1 are similar to that for the Tanner  $[1055, 424]$  code. We observe that  $\mu \in [1, 3.3]$  and  $\rho = 1.9$  are good parameter choices for the PEG  $[2048, 1018]$  code.





**Figure 12:** Choice of embedding method: WER plotted as a function of ADMM step size  $\mu$  for the Tanner [1055, 424] code in  $\mathbb{F}_2$ . The SNR is  $E_s/N_0 = 5\text{dB}$  and  $T_{\max} = 200$ . “CW” stands for “constant weight”.

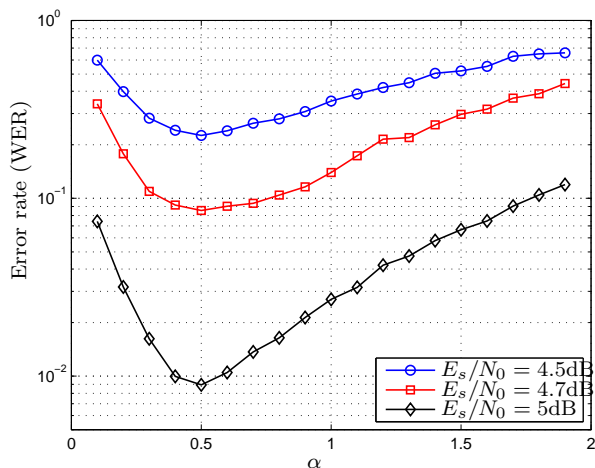


**Figure 13:** Choice of embedding method: Number of iterations plotted as a function of ADMM step size  $\mu$  for the Tanner [1055, 424] code in  $\mathbb{F}_2$ . The SNR is  $E_s/N_0 = 5\text{dB}$  and  $T_{\max} = 200$ . “CW” stands for “constant weight”.

### 7.3 Parameter choices for penalized decoding

We now study the ADMM penalized decoder and show how to choose a good penalty parameter  $\alpha$ . We use the same Tanner [1055, 424] code from Section 7.1 and simulate for the AWGN channel. For this simulation, we do not perform an extensive grid search over the possible parameter choices because of limited computational resources. However, the following parameters are good enough to produce good decoding results:  $\mu = 4$ ,  $\rho = 1.5$ , and  $T_{\max} = 100$ .

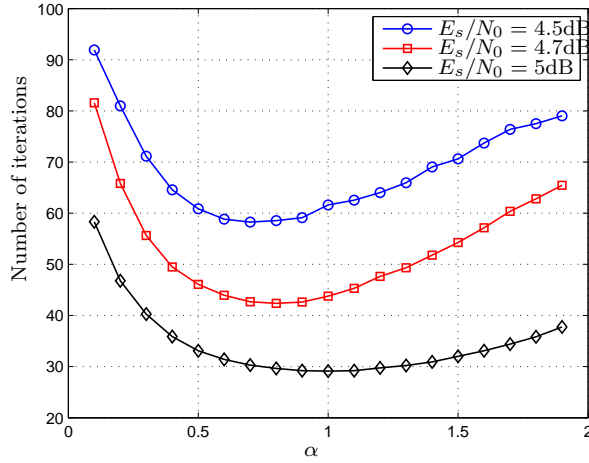
In Figure 14 we plot WER as a function of the penalty parameter  $\alpha$ . We observe that the WER first decreases as the value of  $\alpha$  increases. But when we penalize too heavily (e.g, when  $\alpha > 0.5$ ), the WER starts to increase again. A similar effect can be observed in Figure 15, i.e., the average number of iterations decreases for  $\alpha$  small, and then starts to increase once  $\alpha$  becomes sufficiently large ( $\alpha > 0.8$  in Figure 15). We make two remarks. First, we observe that  $\alpha = 0.5$  is optimal in terms of WER for all SNRs we simulated. This is **not** necessarily the case for other codes or other SNRs. We do observe that for some codes (e.g., the Tanner [755, 334] code studied in Section 7.1) the optimal value for  $\alpha$  is a (non-constant) function of SNR. Second, we observe that the optimal  $\alpha$  in terms of WER is **not** the optimal in terms of the average number of iterations. As a result, there is a trade-off between the optimal computational complexity and the optimal WERs in choosing  $\alpha$ .



**Figure 14:** Penalty coefficient  $\alpha$ : ADMM penalized decoding WER plotted as a function of  $\alpha$  for the Tanner [1055, 424] code in  $\mathbb{F}_2$ .

## 8 Conclusions

In this paper, we develop two types of ADMM decoders for decoding non-binary codes in fields of characteristic two. The first is a natural extension of the binary ADMM decoder introduced by Barman *et al.* in [12]. This ADMM algorithm requires a sub-routine that projects a vector onto the polytope formed by the embeddings of an SPC code. The second ADMM algorithm leverages the factor graph representation of the  $\mathbb{F}_{2^m}$  SPC code embedding and does not require the projection



**Figure 15:** Penalty coefficient  $\alpha$ : ADMM penalized decoding number of iterations plotted as a function of  $\mu$  for the Tanner  $[1055, 424]$  code in  $\mathbb{F}_{2^2}$ .

sub-routine that is necessary for the first ADMM algorithm. We further improve the efficiency of this latter algorithm using new properties of the embedding of SPC codes that we have discovered.

We summarize three important future directions relevant to this topic. First, the relaxed code polytope considered in this work is conjectured to be tight for  $\mathbb{F}_{2^2}$ . The proof of this conjecture is still open. To the best of the authors' knowledge, there has been no known characterization of  $\mathbb{F}_{2^2}$  pseudocodewords. Therefore we believe this conjecture, if proved, is an important theoretical result to develop en-route to understanding pseudocodewords of non-binary LP decoding. Second, as mentioned many times in the paper, improving the projection sub-routine of ADMM decoding is important future work (i.e., for the “first” type of ADMM algorithm as per the previous paragraph). Finally, we show through execution time statistics that the proposed ADMM LP decoder is efficient and could be interesting in practical (hardware) implementation. However, many missing pieces remain to be developed before the decoder can be applied in practice. In particular, there have been no approximation algorithms developed for ADMM LP decoding that are analogous to the relationship between the min-sum algorithm and the sum-product algorithm. Such algorithms may substantially reduce computational complexity and hence are important directions of future work. Furthermore, studying finite precision effects on ADMM LP decoding is another crucial direction to pursue, one that is very important to hardware implementation.

## Acknowledgments

The authors would like to thank Vitaly Skachek for useful discussions and references. The authors would also like to thank the Associate Editor, Prof. Roxana Smarandache, and three anonymous reviewers for their comments and suggestions.

## Appendix

### 9 Definitions and Lemmas for the constant-weight embedding

In this section, we state the definitions and lemmas for the constant-weight embedding that are omitted in Section 3.

**Definition 34** *Under the constant-weight embedding let  $q = 2^m$  and denote by  $\mathcal{E}'$  the set of valid embedded matrices defined by check  $\mathbf{h}$ . An  $q \times d_c$  binary matrix  $\mathbf{F} \in \mathcal{E}'$  if and only if it satisfies the following conditions:*

- (a)  $f_{ij} \in \{0, 1\}$ , where  $i = 0, \dots, 2^m - 1$  and  $j = 1, \dots, d_c$ .
- (b)  $\sum_{i=0}^{q-1} f_{ij} = 1$  where  $q = 2^m$ .
- (c) For any non-zero  $h \in \mathbb{F}_{2^m}$  and  $k \in [m]$ , let  $\mathcal{B}(k, h) := \{\alpha | \mathbf{b}(h\alpha)_k = 1, \alpha \in \mathbb{F}_{2^m}\}$ , where  $\cdot_k$  denotes the  $k$ -th entry of the vector. Let  $g_j^k = \sum_{i \in \mathcal{B}(k, h_j)} f_{ij}$ , then

$$\sum_{j=1}^{d_c} g_j^k = 0 \quad \text{for all } k \in [m] \quad (9.1)$$

where the addition is in  $\mathbb{F}_2$ .

Note that condition (c) is the same for both embeddings. This is due to two facts. First,  $0 \notin \mathcal{B}(k, h)$ . Second, the constant-weight embedding is indexed starting from 0 (cf. Definition 4). Thus, entries from Definition 7 that participate in  $\mathcal{B}(k, h)$  remain unchanged.

**Lemma 35** *In  $\mathbb{F}_{2^m}$ , let  $\mathcal{C}$  be the set of codewords that correspond to check  $\mathbf{h} = (h_1, h_2, \dots, h_{d_c})$ . Then*

$$\mathbf{F}'(\mathcal{C}) = \mathcal{E}', \quad (9.2)$$

where  $\mathcal{E}'$  is defined by Definition 34.

**Proof** See Appendix 10.2. ■

**Definition 36** *The conditions in Definition 34 can be represented using factor graphs. In this graph, there are three types of nodes: (i) variable nodes, (ii) parity-check nodes and (iii) one-on-check node. Each variable node corresponds to an entry  $f_{ij}$  for  $0 \leq i \leq 2^m - 1$  and  $1 \leq j \leq d_c$ . Each parity-check node corresponds to a parity-check of the  $k$ -th bit where  $k \in [m]$ ; it connects all variable nodes in  $\mathcal{B}(k, h_j)$  for all  $j \in [d_c]$ . Each one-on-check node corresponds to a constraint  $\sum_{i=0}^{q-1} f_{ij} = 1$ .*

**Lemma 37** *Consider the constant-weight embedding and let  $\mathcal{F}'$  be the set of  $q \times d_c$  binary matrices defined by the first two conditions of Definition 34 but with the third condition replaced by the condition  $(c^*)$  defined below, then  $\mathcal{F}' = \mathcal{E}'$ . The complete set of conditions are*

- (a)  $f_{ij} \in \{0, 1\}$ .
- (b)  $\sum_{i=0}^{q-1} f_{ij} = 1$  where  $q = 2^m$ .

(c\*) Let  $\mathcal{K}$  be any nonempty subset of  $[m]$ . For any non-zero  $h \in \mathbb{F}_{2^m}$ , let  $\tilde{\mathcal{B}}(\mathcal{K}, h) := \{\alpha \mid \sum_{k \in \mathcal{K}} \mathbf{b}(h\alpha)_k = 1\}$ , where  $\mathbf{b}(h\alpha)_k$  denotes the  $k$ -th entry of the vector  $\mathbf{b}(h\alpha)$ . Let  $g_j^\mathcal{K} = \sum_{i \in \tilde{\mathcal{B}}(\mathcal{K}, h_j)} f_{ij}$ , then  $\sum_{j=1}^{d_c} g_j^\mathcal{K} = 0$  for all  $\mathcal{K} \subset [m]$ ,  $\mathcal{K} \neq \emptyset$ , where the addition is in  $\mathbb{F}_2$ .

**Proof** See Appendix 10.3 ■

**Definition 38** Let  $\mathcal{C}$  be a non-binary SPC code defined by check vector  $\mathbf{h}$ . Denote by  $\mathbb{V}'$  the “tight code polytope” for the constant-weight embedding where

$$\mathbb{V}' = \text{conv}(\mathbf{F}'(\mathcal{C})).$$

In  $\mathbb{F}_{2^m}$  denote by  $\mathbb{U}'$  the “relaxed code polytope” for the constant-weight embedding where a  $q \times d_c$  matrix  $\mathbf{F} \in \mathbb{U}'$  if and only if the following constraints hold:

- (a)  $f_{ij} \in [0, 1]$ .
- (b)  $\sum_{i=0}^{q-1} f_{ij} = 1$ .
- (c) Let  $\tilde{\mathcal{B}}(\mathcal{K}, h)$  be the same set defined in Lemma 37. Let  $\mathbf{g}^\mathcal{K}$  be a vector of length  $d_c$  such that  $g_j^\mathcal{K} = \sum_{i \in \tilde{\mathcal{B}}(\mathcal{K}, h_j)} f_{ij}$ , then

$$\mathbf{g}^\mathcal{K} \in \mathbb{P}_{d_c} \quad \text{for all } \mathcal{K} \subset [m] \text{ and } \mathcal{K} \neq \emptyset \quad (9.3)$$

**Definition 39** Let  $h$  be a non-zero element in  $\mathbb{F}_q$ . Let  $\mathbf{D}'(q, h)$  be a  $q \times q$  permutation matrix indexed starting from row 0 and column 0, and satisfying the following equation

$$\mathbf{D}'(q, h)_{ij} = \begin{cases} 1 & \text{if } i \cdot h = j, \\ 0 & \text{otherwise,} \end{cases}$$

where the multiplication  $i \cdot h$  is in  $\mathbb{F}_q$ . For a vector  $\mathbf{h} \in \mathbb{F}_q^{d_c}$  such that  $h_j \neq 0$  for all  $j \in [d_c]$ , let

$$\mathbf{D}'(q, \mathbf{h}) = \text{diag}(\mathbf{D}'(q, h_1), \dots, \mathbf{D}'(q, h_{d_c})).$$

We note that Lemma 19, Lemma 20, Lemma 21 and Lemma 22 all hold for the constant-weight embedding when  $\mathbf{D}'$  is used. The proofs for the constant-weight embedding can be easily extended from the proofs for Flanagan’s embedding.

## 10 Proofs

### 10.1 Proof of Proposition 9

First,  $|\mathcal{B}(k, 1)| = 2^{m-1}$  because the  $k$ -th bit is fixed to 1. Since dividing by a fixed  $h$  only permutes the elements in finite fields,  $|\mathcal{B}(k, h)|$  is also  $2^{m-1}$ .

## 10.2 Proof of Lemma 8 and Lemma 35

### Proof of Lemma 8

We first prove that for a vector  $\mathbf{c} \in \mathbb{F}_{2^m}^{d_c}$  and its embedded matrix  $\mathbf{F}$ ,  $g_j^k = 1$  if and only if  $\mathbf{b}(h_j c_j)_k = 1$  for all  $j$  and  $k$ . For a fixed  $j$ , if  $\mathbf{b}(h_j c_j)_k = 1$ , then  $c_j \in \mathcal{B}(k, h_j)$  by the definition of  $\mathcal{B}(k, h_j)$ . Note that the  $j$ -th column of  $\mathbf{F}$ , denoted by  $\mathbf{f}_j$ , is a vector such that  $f_{c_j j} = 1$  and  $f_{ij} = 0$  for all  $i \neq c_j$ . Hence  $(g_j^k :=) \sum_{i \in \mathcal{B}(k, h_j)} f_{ij} = f_{c_j j} = 1$ . If  $\mathbf{b}(h_j c_j)_k = 0$ , then  $c_j \notin \mathcal{B}(k, h_j)$ . Since  $f_{ij} = 0$  for all  $i \neq c_j$ , we get  $\sum_{i \in \mathcal{B}(k, h_j)} f_{ij} = 0$ .

Using this result we show  $\mathbf{F} \in \mathcal{F}(\mathcal{C})$  implies  $\mathbf{F} \in \mathcal{E}$ . If  $\mathbf{F} \in \mathcal{F}(\mathcal{C})$ , then there exists a vector  $\mathbf{c} \in \mathcal{C}$  such that  $\mathbf{F} = \mathbf{F}(\mathbf{c})$ . By the definition of  $\mathbf{F}(\mathbf{c})$ , condition (a) and (b) per Definition 7 are satisfied by  $\mathbf{F}$ . For condition (c), note that  $\mathbf{c}$  is a codeword. This implies that  $v := \sum_{j=1}^{d_c} h_j c_j = 0$  in  $\mathbb{F}_{2^m}$ . Further, due to the fact that we are working in extension fields of 2,  $\mathbf{b}(v)$  is an all-zero vector of length  $m$ . This means that for all  $k = 1, \dots, m$ ,  $\mathbf{b}(v)_k = \sum_{j=1}^{d_c} \mathbf{b}(h_j c_j)_k = 0$ . Thus,  $\sum_{j=1}^{d_c} g_j^k = \sum_{j=1}^{d_c} \mathbf{b}(h_j c_j)_k = 0$ . This implies that condition (c) is satisfied.

Conversely, let  $\mathbf{F} \in \mathcal{E}$ . Then by condition (a) and (b), we know that there is a unique vector  $\mathbf{c} \in \mathbb{F}_{2^m}^{d_c}$  such that  $\mathbf{F} = \mathbf{F}(\mathbf{c})$ . By condition (c), we know that for any fixed  $k$ ,  $\sum_{j=1}^{d_c} \sum_{i \in \mathcal{B}(k, h_j)} f_{ij} = \sum_{j=1}^{d_c} \mathbf{b}(h_j c_j)_k = 0$ . Therefore,  $\mathbf{b}(v)_k = 0$  for all  $k$ , where  $v := \sum_{j=1}^{d_c} h_j c_j$ . This implies that  $\mathbf{c}$  is a valid codeword, or equivalently,  $\mathbf{c} \in \mathcal{C}$  and hence  $\mathbf{F} \in \mathcal{F}(\mathcal{C})$ .

### Proof of Lemma 35

This is a simple consequence due to the bijective relationship between the constant-weight embedding and Flanagan's embedding. Formally, if a matrix  $\mathbf{F}' \in \mathcal{E}'$ , we construct  $\mathbf{F}$  by eliminating the first row of  $\mathbf{F}'$ . Then it is simple to verify that  $\mathbf{F} \in \mathcal{E}$ . By Lemma 8,  $\mathbf{F}$  is the embedding for some codeword  $\mathbf{c}$ . This shows that  $\mathbf{F}' \in \mathcal{F}'(\mathcal{C})$ . Conversely, for each codeword  $\mathbf{c} \in \mathcal{C}$ , we can reverse the process and find an  $\mathbf{F}'$  that satisfies Definition 34.

## 10.3 Proof of Lemma 12 and Lemma 37

### Proof of Lemma 12

We first show that  $\sum_{k \in \mathcal{K}} \mathbf{b}(h_j c_j)_k = 1$  if and only if  $g_j^{\mathcal{K}} = 1$ . Suppose  $g_j^{\mathcal{K}} = 1$ , then there exists a unique  $i$  such that  $i \in \tilde{\mathcal{B}}(\mathcal{K}, h_j)$  and  $f_{ij} = 1$ . Since  $f_{c_j j} = 1$  and condition (b) holds,  $i = c_j$ . Therefore  $c_j \in \tilde{\mathcal{B}}(\mathcal{K}, h_j)$ . By the definition of  $\tilde{\mathcal{B}}(\mathcal{K}, h_j)$ ,  $\sum_{k \in \mathcal{K}} \mathbf{b}(h_j c_j)_k = 1$ . On the other hand, if  $\sum_{k \in \mathcal{K}} \mathbf{b}(h_j c_j)_k = 1$ , then  $c_j \in \tilde{\mathcal{B}}(\mathcal{K}, h_j)$ . This implies  $g_j^{\mathcal{K}} := \sum_{i \in \tilde{\mathcal{B}}(\mathcal{K}, h_j)} f_{ij} = 1$ . Next, note that condition (c\*) only adds more constraints to Definition 7. Thus  $\mathcal{F} \subset \mathcal{E}$ . Conversely, using the same definition in the proof in Appendix 10.2,  $\mathbf{b}(v)_k = \sum_{j=1}^{d_c} \mathbf{b}(h_j c_j)_k = 0$ . This means that for any subset  $\mathcal{K}$ ,  $\sum_{k \in \mathcal{K}} (\sum_{j=1}^{d_c} \mathbf{b}(h_j c_j)_k) = 0$ . In addition,

$$\sum_{k \in \mathcal{K}} \sum_{j=1}^{d_c} \mathbf{b}(h_j c_j)_k = \sum_{j=1}^{d_c} \sum_{k \in \mathcal{K}} \mathbf{b}(h_j c_j)_k.$$

Using the result we just proved, we deduce that  $\sum_{j=1}^{d_c} g_j^{\mathcal{K}} = 0$ . This shows that  $\mathcal{E} \subset \mathcal{F}$ . In other words, we have shown that any valid codeword  $\mathbf{c} \in \mathcal{C}$  satisfies condition (c\*) in Lemma 12.

## Proof of Lemma 37

It is easy to verify this lemma following the same logic in Appendix 10.2.

## 10.4 Proof of Lemma 13

First consider all binary vectors of length  $|\mathcal{K}|$  that sum up to 1. The number of such vectors is  $2^{|\mathcal{K}|-1}$ . The number of binary vectors of length  $m - |\mathcal{K}|$  is  $2^{m-|\mathcal{K}|}$ . Thus  $|\tilde{\mathcal{B}}(\mathcal{K}, h)| = 2^{|\mathcal{K}|-1} \times 2^{m-|\mathcal{K}|} = 2^{m-1}$ .

## 10.5 Proof of Proposition 16

We first introduce notation that we use in this proof. Let “ $\prec$ ” denote the relationship of majorization. Let  $\mathbf{a}$  and  $\mathbf{b}$  be length- $n$  vectors, then  $\mathbf{b}$  majorizes  $\mathbf{a}$  is written as  $\mathbf{a} \prec \mathbf{b}$ . We refer readers to Definition 1 in [12] for the definition of majorization. We use Theorem 1 in [12] in this proof.

Let  $\mathbf{y} \in \mathbb{P}_s$  and consider the length- $st$  vector  $\mathbf{y}^* = (\mathbf{y}, \mathbf{0}_{1 \times (st-s)})$ . First, we show  $\mathbf{y}^* \in \mathbb{P}_{st}$ . This immediately implies that all permutations of  $\mathbf{y}^*$  are in  $\mathbb{P}_{st}$ . Then we show  $\mathbf{x} \prec \mathbf{y}^*$ . By Theorem 1 in [12], this implies that  $\mathbf{x}$  is in the convex hull of permutations of  $\mathbf{y}^*$ , each of which is in  $\mathbb{P}_{st}$ . Therefore  $\mathbf{x} \in \mathbb{P}_{st}$ .

Since  $\mathbf{y} \in \mathbb{P}_s$ , there exists a set  $\mathcal{R}$  of length- $s$  binary vectors with even parity such that  $\mathbf{y} = \sum_{i=1}^{|\mathcal{R}|} w_i \mathbf{r}_i$  for some  $w_i \geq 0$  and  $\sum_{i=1}^{|\mathcal{R}|} w_i = 1$ . Let  $\mathbf{r}_i^* = (\mathbf{r}_i, \mathbf{0}_{1 \times (st-s)})$ , then  $\mathbf{r}_i^*$  is a binary vector of length  $st$  with even parity. Also note that  $\mathbf{y}^* = \sum_{i=1}^{|\mathcal{R}|} w_i \mathbf{r}_i^*$ . This implies  $\mathbf{y}^* \in \mathbb{P}_{st}$ .

In order to show  $\mathbf{x} \prec \mathbf{y}^*$ , we need to consider partial sums based on the sorted vectors of  $\mathbf{x}$  and  $\mathbf{y}^*$ . Let  $\mathcal{Q}_k$  (resp.  $\mathcal{U}_k$ ) be the set indices of the  $k$  largest entries in  $\mathbf{x}$  (resp.  $\mathbf{y}^*$ ). Let the operator  $\text{cover}(\cdot)$  be defined as  $i := \text{cover}(j)$  if  $j \in \{(i-1)s+1, (i-1)s+2, \dots, is\}$ . Since all entries of  $\mathbf{x}$  and  $\mathbf{y}^*$  are non-negative,  $x_j \leq y_{\text{cover}(j)}^*$  for all  $j$ . Therefore  $\sum_{j \in \mathcal{Q}_k} x_j \leq \sum_{j \in \mathcal{Q}_k} y_{\text{cover}(j)}^* \leq \sum_{i \in \mathcal{U}_k} y_i^*$ , where the second inequality is because  $\mathcal{U}_k$  contains the largest  $k$  entries of  $\mathbf{y}^*$ . In addition,  $\|\mathbf{x}\|_1 = \|\mathbf{y}^*\|_1$ . By the definition of majorization, these imply that  $\mathbf{x} \prec \mathbf{y}^*$ .

## 10.6 Proof of lemmas on rotation

### 10.6.1 Proof of Lemma 19

We first show that if  $\mathbf{u}^N \in \mathbb{V}^N$  then  $\mathbf{u} = \mathbf{D}(q, \mathbf{h})\mathbf{u}^N \in \mathbb{V}$ . Let  $\mathbf{e}_k^N$ ,  $k = 1, \dots, |\mathcal{E}^N|$ , be vectors in  $\mathcal{E}^N$ , then

$$\begin{aligned} \mathbf{D}(q, \mathbf{h})\mathbf{u}^N &= \mathbf{D}(q, \mathbf{h}) \sum_k \alpha_k \mathbf{e}_k^N \\ &= \sum_k \alpha_k (\mathbf{D}(q, \mathbf{h}) \mathbf{e}_k^N) \\ &= \sum_k \alpha_k \mathbf{e}_k, \end{aligned}$$

where  $\mathbf{e}_k$  are vectors in  $\mathcal{E}$ . Therefore  $\mathbf{v} \in \mathbb{V}$ .

Conversely, if  $\mathbf{u} \in \mathbb{V}$ ,  $\mathbf{u}^N = \mathbf{D}(q, \mathbf{h})^{-1} \mathbf{u}$ . Let  $\mathbf{e}_k$ ,  $k = 1, \dots, |\mathcal{E}|$ , be vectors in  $\mathcal{E}$ , then

$$\begin{aligned} \mathbf{D}(q, \mathbf{h})^{-1} \mathbf{u} &= \mathbf{D}(q, \mathbf{h})^{-1} \sum_k \alpha_k \mathbf{e}_k \\ &= \sum_k \alpha_k (\mathbf{D}(q, \mathbf{h})^{-1} \mathbf{e}_k) \\ &= \sum_k \alpha_k \mathbf{e}_k^N, \end{aligned}$$

where  $\mathbf{e}_k^N$  are vectors in  $\mathcal{E}^N$ . Therefore  $\mathbf{u} \in \mathbb{V}^N$ .

### 10.6.2 Proof of Lemma 20

First, by Lemma 19,  $\mathbf{u} \in \mathbb{V}$ . Therefore we need to show that there is no vector in  $\mathbb{V}$  that has a smaller Euclidean distance to  $\mathbf{v}$  than does  $\mathbf{u}$ . Suppose that there is a vector  $\mathbf{p}$  such that  $\|\mathbf{p} - \mathbf{v}\|_2^2 < \|\mathbf{D}(q, \mathbf{h})\mathbf{u}^N - \mathbf{v}\|_2^2$ . Then  $\|\mathbf{p} - \mathbf{v}\|_2^2 = \|\mathbf{D}(q, \mathbf{h})^{-1}(\mathbf{p} - \mathbf{v})\|_2^2 = \|\mathbf{D}(q, \mathbf{h})^{-1}\mathbf{p} - \mathbf{D}(q, \mathbf{h})^{-1}\mathbf{v}\|_2^2 = \|\mathbf{D}(q, \mathbf{h})^{-1}\mathbf{p} - \mathbf{v}^N\|_2^2$ , where the second equality is due to the fact that  $\mathbf{D}(q, \mathbf{h})$  is a permutation matrix. We then deduce that

$$\|\mathbf{D}(q, \mathbf{h})^{-1}\mathbf{p} - \mathbf{v}^N\|_2^2 < \|\mathbf{u}^N - \mathbf{D}(q, \mathbf{h})^{-1}\mathbf{v}\|_2^2,$$

which contradicts with the fact that  $\mathbf{u}^N$  is the projection of  $\mathbf{v}^N$  onto  $\mathbb{V}^N$ .

### 10.6.3 Proof of Lemma 21

Let  $\mathbf{w} \in \mathbb{U}^N$ , we need to show  $\mathbf{u} := \mathbf{D}(q, \mathbf{h})\mathbf{w}$  is in  $\mathbb{U}$ . Let  $\mathbf{U}$  and  $\mathbf{W}$  be the matrix form for  $\mathbf{u}$  and  $\mathbf{w}$  respectively (i.e.  $\text{vec}(\mathbf{U}) = \mathbf{u}$  and  $\text{vec}(\mathbf{W}) = \mathbf{w}$ ). Note that the rotations are done per embedded vector. In other words, columns of  $\mathbf{U}$  are permuted independently of each other. By the definition of the rotation matrix, for each column  $j$ ,

$$u_{ij} = w_{lj} \text{ for all } l = i \cdot h, \quad (10.1)$$

where the multiplication is in  $\mathbb{F}_{2^m}$ . Since  $\mathbf{w} \in \mathbb{U}^N$ ,  $\mathbf{W}$  satisfies Definition 15 for check  $(1, 1, 1, \dots, 1)$ . We first verify that  $\mathbf{U}$  satisfies conditions (a) and (b) in Definition 15. Note that these two conditions are defined on a column-wise basis. Thus these two conditions are invariant to column-wise permutations. For condition (c), note that  $\tilde{\mathcal{B}}(\mathcal{K}, h)$  can be obtained by  $\tilde{\mathcal{B}}(\mathcal{K}, 1)$  as follows:

$$\tilde{\mathcal{B}}(\mathcal{K}, h) = \{y | y = x/h, x \in \tilde{\mathcal{B}}(\mathcal{K}, 1)\}. \quad (10.2)$$

Then,  $i \in \tilde{\mathcal{B}}(\mathcal{K}, h_j)$  if  $l \in \tilde{\mathcal{B}}(\mathcal{K}, 1)$ , provided that  $l = i \cdot h_j$ . Let  $\mathbf{f}_{\tilde{\mathcal{B}}(\mathcal{K}, h_j), j}$  be the sub-vector constructed by selecting entries  $f_{ij}$  where  $i \in \tilde{\mathcal{B}}(\mathcal{K}, h_j)$ . Combining (10.1) and (10.2), we deduce that for fixed  $j$ ,  $\mathcal{K}$  and  $l$ , if  $w_{lj}$  is in the vector  $\mathbf{f}_{\tilde{\mathcal{B}}(\mathcal{K}, 1), j}$ , then  $u_{ij}$  is in the vector  $\mathbf{f}_{\tilde{\mathcal{B}}(\mathcal{K}, h_j), j}$  and  $u_{ij} = w_{lj}$ . This shows that  $g_j^\mathcal{K} = \|\mathbf{f}_{\tilde{\mathcal{B}}(\mathcal{K}, h_j), j}\|_1$  is not changed under rotation. Therefore  $\mathbf{u}$  satisfies condition (c). This implies that  $\mathbf{u} \in \mathbb{U}$ .

Conversely, let  $\mathbf{u} \in \mathbb{U}$ , we need to show  $\mathbf{w} := \mathbf{D}(q, \mathbf{h})^{-1}\mathbf{u}$  is in  $\mathbb{U}^N$ . Note that  $\mathbf{D}(q, \mathbf{h})$  is a permutation matrix and therefore  $\mathbf{D}(q, \mathbf{h})^{-1} = \mathbf{D}(q, \mathbf{h})^T$ . As in the previous case, the columns of  $\mathbf{U}$  and  $\mathbf{W}$  are permuted independently. Therefore condition (a) and (b) hold for  $\mathbf{W}$ . We now verify



condition (c). First, permutation using  $\mathbf{D}(q, \mathbf{h})^T$  implies that  $u_{ij} = w_{lj}$  for all  $l = i \cdot h$ . Second,  $\tilde{\mathcal{B}}(\mathcal{K}, 1)$  can be obtained by  $\tilde{\mathcal{B}}(\mathcal{K}, h)$  as follows:

$$\tilde{\mathcal{B}}(\mathcal{K}, 1) = \{y | y = x \cdot h, x \in \tilde{\mathcal{B}}(\mathcal{K}, h)\}. \quad (10.3)$$

Thus, for fixed  $j$ ,  $\mathcal{K}$  and  $l$ , if entry  $u_{ij}$  is contained in the vector  $\mathbf{f}_{\tilde{\mathcal{B}}(\mathcal{K}, h_j), j}$ , then entry  $w_{lj}$  is contained in the vector  $\mathbf{f}_{\tilde{\mathcal{B}}(\mathcal{K}, 1), j}$  and  $u_{ij} = w_{lj}$ . Therefore the vector  $\mathbf{g}^{\mathcal{K}}$  remains unchanged, which implies that condition (c) is satisfied by  $\mathbf{W}$ .

#### 10.6.4 Proof of Lemma 22

Using Lemma 21, we can follow exactly the same logic as in the proof of Lemma 21. The details are omitted.

### 10.7 Proof of Theorem 24

We first prove several lemmas that will be useful for proving the theorem.

**Lemma 40** For any  $\alpha \in \mathbb{F}_q$ ,  $\mathbf{f}'(\alpha) = (1 - \|\mathbf{f}(\alpha)\|_1, \mathbf{f}(\alpha))$ .

**Proof** If  $\alpha \neq 0$ ,  $\|\mathbf{f}(\alpha)\|_1 = 1$  and  $\mathbf{f}'(\alpha) = (0, \mathbf{f}(\alpha))$ . If  $\alpha = 0$ ,  $\|\mathbf{f}(\alpha)\|_1 = 0$  and  $\mathbf{f}'(\alpha) = (1, 0, 0, \dots, 0)$ .  $\blacksquare$

**Lemma 41** Let  $\mathcal{C}$  be the SPC code defined by check  $\mathbf{h}$ . If

$$\hat{\mathbf{f}} = (\mathbf{f}_1 | \dots | \mathbf{f}_{d_c})^T \in \text{conv}(\mathbf{F}_v(\mathcal{C})),$$

then

$$\bar{\mathbf{f}} = (a_1, \mathbf{f}_1 | \dots | a_{d_c}, \mathbf{f}_{d_c})^T \in \text{conv}(\mathbf{F}'_v(\mathcal{C})),$$

where  $a_i = 1 - \|\mathbf{f}_i\|_1$  for all  $i = 1, \dots, d_c$ . Conversely, if

$$\bar{\mathbf{f}} = (a_1, \mathbf{f}_1 | \dots | a_{d_c}, \mathbf{f}_{d_c})^T \in \text{conv}(\mathbf{F}'_v(\mathcal{C}))$$

for some  $a_1, \dots, a_{d_c}$ , then

$$\hat{\mathbf{f}} = (\mathbf{f}_1 | \dots | \mathbf{f}_{d_c})^T \in \text{conv}(\mathbf{F}_v(\mathcal{C})).$$

**Proof** Let  $\mathbf{c}^k$ ,  $k = 1, \dots, |\mathcal{C}|$  be all codewords in  $\mathcal{C}$ . Let  $\hat{\mathbf{f}} \in \text{conv}(\mathbf{F}_v(\mathcal{C}))$ , then there exists a set of coefficients  $0 \leq \omega_k \leq 1$ ,  $k = 1, \dots, |\mathcal{C}|$ , satisfying  $\sum_{k=1}^{|\mathcal{C}|} \omega_k = 1$  such that  $\hat{\mathbf{f}} = \sum_{k=1}^{|\mathcal{C}|} \omega_k \mathbf{F}_v(\mathbf{c}^k)$ . Then,  $\mathbf{f}_i = \sum_{k=1}^{|\mathcal{C}|} \omega_k \mathbf{f}_i(\mathbf{c}^k)$  for all  $i$ . On the other hand,  $\sum_{k=1}^{|\mathcal{C}|} \omega_k \mathbf{F}'_v(\mathbf{c}^k) \in \text{conv}(\mathbf{F}'_v(\mathcal{C}))$  by the definition of convex hull. Using Lemma 40,

$$\begin{aligned} \sum_{k=1}^{|\mathcal{C}|} \omega_k \mathbf{F}'_v(\mathbf{c}^k) &= \left( \sum_{k=1}^{|\mathcal{C}|} \omega_k \mathbf{f}'(\mathbf{c}_1^k), \dots, \sum_{k=1}^{|\mathcal{C}|} \omega_k \mathbf{f}'(\mathbf{c}_{d_c}^k) \right) \\ &= \left( \sum_{k=1}^{|\mathcal{C}|} \omega_k (1 - \|\mathbf{f}(\mathbf{c}_1^k)\|_1, \mathbf{f}(\mathbf{c}_1^k)), \dots, \sum_{k=1}^{|\mathcal{C}|} \omega_k (1 - \|\mathbf{f}(\mathbf{c}_{d_c}^k)\|_1, \mathbf{f}(\mathbf{c}_{d_c}^k)) \right) \\ &= (1 - \|\mathbf{f}_1\|_1, \mathbf{f}_1, \dots, 1 - \|\mathbf{f}_{d_c}\|_1, \mathbf{f}_{d_c}) = \bar{\mathbf{f}}. \end{aligned}$$

Thus we conclude  $\bar{\mathbf{f}} \in \text{conv}(\mathbf{F}'_v(\mathcal{C}))$ .

Conversely, if  $\bar{\mathbf{f}} \in \text{conv}(\mathbf{F}'_v(\mathcal{C}))$ , then there exists a set of coefficients  $0 \leq \omega_k \leq 1$ ,  $k = 1, \dots, |\mathcal{C}|$ , satisfying  $\sum_{k=1}^{|\mathcal{C}|} \omega_k = 1$  such that  $\bar{\mathbf{f}} = \sum_{k=1}^{|\mathcal{C}|} \omega_k \mathbf{F}'_v(\mathbf{c}^k)$ . By definition,  $\sum_{k=1}^{|\mathcal{C}|} \omega_k \mathbf{F}_v(\mathbf{c}^k) \in \text{conv}(\mathbf{F}_v(\mathcal{C}))$ . Similar to the previous case, it is easy to verify using Lemma 40 that  $\sum_{k=1}^{|\mathcal{C}|} \omega_k \mathbf{F}_v(\mathbf{c}^k) = \hat{\mathbf{f}}$ . ■

**Lemma 42** *If  $\hat{\mathbf{f}} = (\mathbf{f}_1 | \dots | \mathbf{f}_N)^T$  is feasible for **LP-FT**, then  $\bar{\mathbf{f}} = (1 - \|\mathbf{f}_1\|_1, \mathbf{f}_1 | \dots | 1 - \|\mathbf{f}_N\|_1, \mathbf{f}_N)^T$  is feasible for **LP-CT**. Conversely, if  $\bar{\mathbf{f}} = (f_1, \mathbf{f}_1 | \dots | f_N, \mathbf{f}_N)^T$  is feasible to **LP-CT**, then  $\hat{\mathbf{f}} = (\mathbf{f}_1 | \dots | \mathbf{f}_N)^T$  is feasible for **LP-FT**.*

**Proof** For LDPC codes, check  $j \in \mathcal{J}$  selects a sub-vector  $\hat{\mathbf{f}}_j$  from  $\hat{\mathbf{f}}$  (or  $\bar{\mathbf{f}}_j$  from  $\bar{\mathbf{f}}$ ) that participates in that check. Therefore we apply Lemma 41 to each sub-vector of  $\hat{\mathbf{f}}$  (and of  $\bar{\mathbf{f}}$ ). In addition,  $\hat{\mathbf{f}}$  (or  $\bar{\mathbf{f}}$ ) is feasible if and only if  $\hat{\mathbf{f}}_j \in \text{conv}(\mathbf{F}_v(\mathcal{C}_j))$  ( $\bar{\mathbf{f}}_j \in \text{conv}(\mathbf{F}'_v(\mathcal{C}_j))$ ) for all  $j \in \mathcal{J}$ , where  $\mathcal{C}_j$  is the SPC code defined by the  $j$ -th check. Therefore we conclude that the lemma holds. ■

**Lemma 43** *Let  $\mathbf{y} \in \Sigma$ ,  $\hat{\mathbf{f}} = (\mathbf{f}_1 | \dots | \mathbf{f}_N)^T$  be a vector with non-negative entries and such that  $\|\mathbf{f}_i\|_1 \leq 1$ . Further, let  $\bar{\mathbf{f}} = (1 - \|\mathbf{f}_1\|_1, \mathbf{f}_1 | \dots | 1 - \|\mathbf{f}_N\|_1, \mathbf{f}_N)^T$ . Then*

$$\mathbf{L}'_v(\mathbf{y})\bar{\mathbf{f}} = \mathbf{L}_v(\mathbf{y})\hat{\mathbf{f}} - \sum_{i=1}^N \log(\Pr[y_i|0]).$$

**Proof** For simplicity, we index entries in  $\hat{\mathbf{f}}$  starting from 0. By definitions of  $\hat{\mathbf{f}}$  and  $\bar{\mathbf{f}}$ , we have

$$\begin{aligned} \mathbf{f}_i &= (\bar{f}_{(i-1)q+1}, \bar{f}_{(i-1)q+2}, \dots, \bar{f}_{(i-1)q+q-1}) \\ &= (\hat{f}_{(i-1)(q-1)+1}, \hat{f}_{(i-1)(q-1)+2}, \dots, \hat{f}_{(i-1)(q-1)+q-1}), \end{aligned}$$

and

$$\bar{f}_{(i-1)q} = 1 - \|\mathbf{f}_i\|_1.$$

Note that  $\bar{f}_{(i-1)q+\delta} = \hat{f}_{(i-1)(q-1)+\delta}$  for all  $i \in \mathcal{I}$  and  $\delta \in [q-1]$ . Then

$$\begin{aligned} \mathbf{L}'_v(\mathbf{y})\bar{\mathbf{f}} &= \sum_{i=1}^N \sum_{\delta=0}^{q-1} \log\left(\frac{1}{\Pr[y_i|\delta]}\right) \bar{f}_{(i-1)q+\delta} \\ &= \sum_{i=1}^N \left[ \sum_{\delta=1}^{q-1} \log\left(\frac{1}{\Pr[y_i|\delta]}\right) \bar{f}_{(i-1)q+\delta} + (1 - \|\mathbf{f}_i\|_1) \log\left(\frac{1}{\Pr[y_i|0]}\right) \right] \\ &= \sum_{i=1}^N \left[ \sum_{\delta=1}^{q-1} \log\left(\frac{\Pr[y_i|0]}{\Pr[y_i|\delta]}\right) \bar{f}_{(i-1)q+\delta} + \log\left(\frac{1}{\Pr[y_i|0]}\right) \right] \\ &= \sum_{i=1}^N \left[ \sum_{\delta=1}^{q-1} \log\left(\frac{\Pr[y_i|0]}{\Pr[y_i|\delta]}\right) \hat{f}_{(i-1)(q-1)+\delta} \right] + \sum_{i=1}^N \log\left(\frac{1}{\Pr[y_i|0]}\right) \\ &= \mathbf{L}_v(\mathbf{y})\hat{\mathbf{f}} - \sum_{i=1}^N \log(\Pr[y_i|0]). \end{aligned}$$

■

Now we state the proof of Theorem 24: **Proof** By Lemma 42,  $\bar{\mathbf{f}} = (a_1, \mathbf{f}_1 | \dots | a_N, \mathbf{f}_N)^T$  is feasible for **LP-CT** if and only if  $\hat{\mathbf{f}} = (\mathbf{f}_1 | \dots | \mathbf{f}_N)^T$  is feasible for **LP-FT**. By Lemma 43, we can subtract a constant,  $\sum_{i=1}^N \log(\Pr[y_i|0])$ , from the objective function of **LP-CT** to obtain the same objective function as **LP-FT**.

We now prove the theorem by contradiction. Suppose  $\bar{\mathbf{f}}$  is the solution of **LP-CT**, but  $\bar{\mathbf{f}}$  is not the solution of **LP-FT**. Then there exists a feasible point  $\tilde{\mathbf{f}}$  that attains a lower cost than  $\bar{\mathbf{f}}$ . Using Lemma 42, we can construct a feasible point for **LP-CT** that attains a fewer cost than  $\bar{\mathbf{f}}$ . This contradicts with the assumption that  $\bar{\mathbf{f}}$  is the solution of **LP-CT**. Similar arguments also hold for the converse statement.  $\blacksquare$

## 10.8 Proof of Theorem 27

We first restate Lemma 41 and Lemma 42 for the constant-weight embedding:

**Lemma 44** *Let  $\mathbb{U}$  (resp.  $\mathbb{U}'$ ) be the relaxed code polytope for Flanagan's embedding (resp. the constant-weight embedding) for check  $\mathbf{h}$ . If  $\hat{\mathbf{f}} = (\mathbf{f}_1 | \dots | \mathbf{f}_{d_c})^T \in \mathbb{U}$ , then  $\bar{\mathbf{f}} = (a_1, \mathbf{f}_1 | \dots | a_{d_c}, \mathbf{f}_{d_c})^T \in \mathbb{U}'$  where  $a_i = 1 - \|\mathbf{f}_i\|_1$  for all  $i = 1, \dots, d_c$ . Conversely, if  $\bar{\mathbf{f}} = (a_1, \mathbf{f}_1 | \dots | a_{d_c}, \mathbf{f}_{d_c})^T \in \mathbb{U}'$  for some  $a_1, \dots, a_{d_c}$ , then  $\hat{\mathbf{f}} = (\mathbf{f}_1 | \dots | \mathbf{f}_{d_c})^T \in \mathbb{U}$ .*

**Proof** For any value of  $i \in [d_c]$ ,  $f_{0i}$  does not participate in the condition (c) in Definition 38. Thus  $\hat{\mathbf{f}}$  satisfies condition (c) in Definition 15 if and only if  $\bar{\mathbf{f}}$  satisfies condition (c) in Definition 38. Further, it is easy to verify that  $\hat{\mathbf{f}}$  satisfies condition (a) and (b) in Definition 15 if and only if  $\bar{\mathbf{f}}$  satisfies condition (a) and (b) in Definition 38.  $\blacksquare$

**Lemma 45** *If  $\hat{\mathbf{f}} = (\mathbf{f}_1 | \dots | \mathbf{f}_N)^T$  is feasible for **LP-FR**, then  $\bar{\mathbf{f}} = (1 - \|\mathbf{f}_1\|_1, \mathbf{f}_1 | \dots | 1 - \|\mathbf{f}_N\|_1, \mathbf{f}_N)^T$  is feasible for **LP-CR**. Conversely, if  $\bar{\mathbf{f}} = (f_1, \mathbf{f}_1 | \dots | f_N, \mathbf{f}_N)^T$  is feasible to **LP-CR**, then  $\hat{\mathbf{f}} = (\mathbf{f}_1 | \dots | \mathbf{f}_N)^T$  is feasible to **LP-FR**.*

**Proof** The proof is the same as for Lemma 42 except it is based on Lemma 44.  $\blacksquare$

We can prove Theorem 27 using Lemma 43 and Lemma 45. The logic is the same as that for Theorem 24.

## 10.9 Proof of Corollary 29

**LP-FR** has the same embedding and objective as **LP-FT**. Thus we can follow the same proof as in [16, Thm. 5.1]. The only difference is in the constraint set. Thus we only need to prove that the “relative matrix” (defined below) to the all-zeros matrix satisfies the constraint set in **LP-FR**. The rest of the proof is identical to that in [16]. The following definition rephrases Eq. (19) in [16].

**Definition 46** *Let  $\mathcal{C}$  be a SPC code defined by a length- $d$  check  $\mathbf{h}$ . Let  $\mathbf{F}$  and  $\tilde{\mathbf{F}}$  be  $(2^m - 1) \times d$  matrices. We say that  $\tilde{\mathbf{F}}$  is the “relative matrix” for  $\mathbf{F}$  based on a codeword  $\mathbf{c} \in \mathcal{C}$  if*

$$\tilde{f}_{ij} = \begin{cases} 1 - \sum_i f_{ij} & \text{if } i + c_j = 0 \\ f_{(i+c_j)j} & \text{otherwise} \end{cases}, \quad (10.4)$$

where the sums are in  $\mathbb{F}_{2^m}$ . (Note that in  $\mathbb{F}_{2^m}$ ,  $p+q=0$  means  $q=p$ .) Denote this mapping  $R_c(\cdot)$ . Note that this is a bijective mapping. Its inverse is given by

$$f_{ij} = \begin{cases} 1 - \sum_i \tilde{f}_{ij} & \text{if } i - c_j = 0 \\ \tilde{f}_{(i-c_j)j} & \text{otherwise} \end{cases}. \quad (10.5)$$

Denote by  $R_c^{-1}(\cdot)$  this inverse mapping.

**Lemma 47** *Let  $c$  be a valid SPC for length- $d$  check  $h$ . Then  $F \in \mathbb{U}$  if and only if  $R_c(F) \in \mathbb{U}$ .*

**Proof**

- We first show that if  $F \in \mathbb{U}$ , then  $R_c(F) \in \mathbb{U}$ . In other words, we need to verify that  $R_c(F)$  satisfies all three conditions in Definition 15. The first two conditions are obvious. We focus on the third condition. Let  $\mathcal{K}$  be a non-empty subset of  $[m]$  and let  $j$  be an integer in  $[d]$ . Let  $\tilde{\mathcal{B}}(\mathcal{K}, h_j)$  be the set defined in Lemma 12 for the  $j$ -th check entry  $h_j$ . There are two cases: (i)  $c_j \notin \tilde{\mathcal{B}}(\mathcal{K}, h_j)$  and (ii)  $c_j \in \tilde{\mathcal{B}}(\mathcal{K}, h_j)$ .

If  $c_j \notin \tilde{\mathcal{B}}(\mathcal{K}, h_j)$  then for all  $i \in \tilde{\mathcal{B}}(\mathcal{K}, h_j)$ ,

$$\begin{aligned} \sum_{k \in \mathcal{K}} b(h_j(i + c_j))_k &= \sum_{k \in \mathcal{K}} b(h_j i + h_j c_j)_k \\ &= \sum_{k \in \mathcal{K}} [b(h_j i)_k + b(h_j c_j)_k] \\ &= 1 + 0 = 1, \end{aligned}$$

where the second equality follows because that the addition in  $\mathbb{F}_{2^m}$  is equivalent to the vector addition of the corresponding binary vectors. The third equality follows because  $i \in \tilde{\mathcal{B}}(\mathcal{K}, h_j)$  and  $c_j \notin \tilde{\mathcal{B}}(\mathcal{K}, h_j)$ . Similarly, for all  $i \notin \tilde{\mathcal{B}}(\mathcal{K}, h_j)$ ,  $\sum_{k \in \mathcal{K}} b(h_j(i + c_j))_k = 0$ . Thus  $i + c_j \in \tilde{\mathcal{B}}(\mathcal{K}, h_j)$  if and only if  $i \in \tilde{\mathcal{B}}(\mathcal{K}, h_j)$ .

In addition, since  $c_j \notin \tilde{\mathcal{B}}(\mathcal{K}, h_j)$ , we find that  $c_j \neq i$  for all  $i \in \tilde{\mathcal{B}}(\mathcal{K}, h_j)$ . Then, by Definition 46,  $\tilde{f}_{ij} = f_{(i+c_j)j}$ . Let  $g_j^\mathcal{K} := \sum_{i \in \tilde{\mathcal{B}}(\mathcal{K}, h_j)} f_{ij}$  and  $\tilde{g}_j^\mathcal{K} := \sum_{i \in \tilde{\mathcal{B}}(\mathcal{K}, h_j)} \tilde{f}_{ij}$ , then

$$\begin{aligned} \sum_{i \in \tilde{\mathcal{B}}(\mathcal{K}, h_j)} \tilde{f}_{ij} &= \sum_{i \in \tilde{\mathcal{B}}(\mathcal{K}, h_j)} f_{(i+c_j)j} \\ &= \sum_{(i+c_j) \in \tilde{\mathcal{B}}(\mathcal{K}, h_j)} f_{(i+c_j)j} \\ &= \sum_{l \in \tilde{\mathcal{B}}(\mathcal{K}, h_j)} f_{lj} = g_j^\mathcal{K}, \end{aligned}$$

where the second equality follows because  $i + c_j \in \tilde{\mathcal{B}}(\mathcal{K}, h_j)$  if and only if  $i \in \tilde{\mathcal{B}}(\mathcal{K}, h_j)$ . Therefore we conclude that  $g_j^\mathcal{K} = \tilde{g}_j^\mathcal{K}$ .

For case (ii),  $c_j \in \tilde{\mathcal{B}}(\mathcal{K}, h_j)$ . Using the same argument as above, we can show that  $i + c_j \in \tilde{\mathcal{B}}(\mathcal{K}, h_j)$  if and only if  $i \notin \tilde{\mathcal{B}}(\mathcal{K}, h_j)$ . Therefore

$$\begin{aligned} \sum_{i \in \tilde{\mathcal{B}}(\mathcal{K}, h_j)} \tilde{f}_{ij} &= \tilde{f}_{c_j j} + \sum_{i \in \tilde{\mathcal{B}}(\mathcal{K}, h_j) \text{ and } i \neq c_j} \tilde{f}_{ij} \\ &= \left(1 - \sum_{i=1}^{2^m-1} f_{ij}\right) + \sum_{l \notin \tilde{\mathcal{B}}(\mathcal{K}, h_j) \text{ and } l \neq 0} f_{lj} \\ &= 1 - \sum_{l \in \tilde{\mathcal{B}}(\mathcal{K}, h_j)} f_{lj} \\ &= 1 - g_j^{\mathcal{K}}. \end{aligned}$$

Combining the two cases, we conclude that the vector  $\tilde{\mathbf{g}}^{\mathcal{K}}$  satisfies the following conditions:

$$\tilde{g}_j^{\mathcal{K}} = \begin{cases} g_j^{\mathcal{K}} & \text{if } c_j \notin \tilde{\mathcal{B}}(\mathcal{K}, h_j) \\ 1 - g_j^{\mathcal{K}} & \text{if } c_j \in \tilde{\mathcal{B}}(\mathcal{K}, h_j) \end{cases}. \quad (10.6)$$

We can rephrase this condition by introducing the following notation: Let  $\tilde{\mathbf{F}} = \mathbf{F}(\mathbf{c})$ . Let  $\mathbf{g}^{\mathcal{K}, \mathbf{c}}$  be a binary vector for  $\mathcal{K}$  and  $\mathbf{c}$  defined by  $g_j^{\mathcal{K}, \mathbf{c}} := \sum_{i \in \tilde{\mathcal{B}}(\mathcal{K}, h_j)} \tilde{f}_{ij}$ . By Lemma 12,  $\mathbf{g}^{\mathcal{K}, \mathbf{c}}$  is a binary vector with even parity. By its definition,  $g_j^{\mathcal{K}, \mathbf{c}} = 1$  if and only if  $c_j \in \tilde{\mathcal{B}}(\mathcal{K}, h_j)$ . Thus we can rewrite (10.6) as

$$\tilde{g}_j^{\mathcal{K}} = \begin{cases} g_j^{\mathcal{K}} & \text{if } g_j^{\mathcal{K}, \mathbf{c}} = 0 \\ 1 - g_j^{\mathcal{K}} & \text{if } g_j^{\mathcal{K}, \mathbf{c}} = 1 \end{cases}. \quad (10.7)$$

These conditions satisfy the definition of “relative solution” defined in [2]. When applying Lemma 17 in [2] to the case of binary single parity-check code, we conclude that  $\tilde{\mathbf{g}}^{\mathcal{K}} \in \mathbb{P}_d$  if  $\mathbf{g}^{\mathcal{K}} \in \mathbb{P}_d$ . This conclude our verification of the third condition of Definition 15.

- Next we need to show that if  $\tilde{\mathbf{F}} \in \mathbb{U}$ , then  $\mathbf{R}_{\mathbf{c}}^{-1}(\tilde{\mathbf{F}}) \in \mathbb{U}$ . Note that in (10.5),  $i - c_j = 0$  is equivalent to  $i + c_j = 0$  for  $\mathbb{F}_{2^m}$ . Therefore the proof is identical to the previous case. ■

## 10.10 Proof of Theorem 33

### 10.10.1 Sketch of the proof

We need to prove  $\Pr[\text{error}|\mathbf{0}] = \Pr[\text{error}|\mathbf{c}]$ , where  $\mathbf{c}$  is any non-zero codeword. Let

$$\mathcal{B}(\mathbf{c}) := \{\mathbf{y} | \text{Decoder fails to recover } \mathbf{c} \text{ if } \mathbf{y} \text{ is received}\}.$$

Then  $\Pr[\text{error}|\mathbf{c}] = \sum_{\mathbf{y} \in \mathcal{B}(\mathbf{c})} \Pr[\mathbf{y}|\mathbf{c}]$ .

We first rephrase the symmetry condition of [16]. This definition introduces a one-to-one mapping from the received vector  $\mathbf{y}$  to a vector  $\mathbf{y}^0$  such that the following two statements hold:

- (a)  $\Pr[\mathbf{y}|\mathbf{c}] = \Pr[\mathbf{y}^0|\mathbf{0}]$ ,

(b)  $\mathbf{y} \in \mathcal{B}(\mathbf{c})$  if and only if  $\mathbf{y}^0 \in \mathcal{B}(\mathbf{0})$ .

Statement (a) is directly implied by the definition of the symmetry condition. We show in the following that statement (b) is also true. Once we have both results,

$$\begin{aligned} \Pr[\text{error}|\mathbf{c}] &= \sum_{\mathbf{y} \in \mathcal{B}(\mathbf{c})} \Pr[\mathbf{y}|\mathbf{c}] \\ &= \sum_{\mathbf{y}^0 \in \mathcal{B}(\mathbf{0})} \Pr[\mathbf{y}^0|\mathbf{c}] = \Pr[\text{error}|\mathbf{0}]. \end{aligned}$$

### 10.10.2 Symmetry condition

A symmetry condition for rings is defined in [16]. We now rephrase that definition for fields.

**Definition 48** (*Symmetry condition*) For  $\beta \in \mathbb{F}_q$ , there exists a bijection

$$\tau_\beta : \Sigma \mapsto \Sigma,$$

such that the channel output probability conditioned on the channel input satisfies

$$P(y|\alpha) = P(\tau_\beta(y)|\alpha - \beta), \quad (10.8)$$

for all  $y \in \Sigma$ ,  $\alpha \in \mathbb{F}_q$ . In addition,  $\tau_\beta$  is isometric in the sense of [16].

### 10.10.3 Proof of statement (b)

We define the concept of relative matrices for the constant-weight embedding.

**Definition 49** Let  $\alpha \in \mathbb{F}_{2^m}$  and let  $\mathbf{x}$  be a vector of length  $2^m$ . We say that  $\tilde{\mathbf{x}}$  is the “relative vector” for  $\mathbf{x}$  based on  $\alpha$  if

$$\tilde{x}_i = x_{i+\alpha}, \quad (10.9)$$

where the sum is in  $\mathbb{F}_{2^m}$ . This mapping is denoted by  $\tilde{\mathbf{x}} = \mathbf{R}'_\alpha(\mathbf{x})$ . Its inverse is given by

$$x_i = \tilde{x}_{i-\alpha}. \quad (10.10)$$

Denote by  $\mathbf{R}'_\alpha^{-1}(\cdot)$  this inverse mapping.

We reuse this notation in the context of non-binary vectors and let  $\mathbf{c} \in \mathbb{F}_{2^m}^d$ . Let  $\mathbf{F}'$  and  $\tilde{\mathbf{F}}'$  be  $2^m \times d$  matrices. We say that  $\tilde{\mathbf{F}}'$  is the “relative matrix” of  $\mathbf{F}'$  based on  $\mathbf{c}$  if for all  $j$

$$\tilde{f}'_{ij} = f'_{(i+\mathbf{c}_j)j}, \quad (10.11)$$

where the sum is in  $\mathbb{F}_{2^m}$ . This mapping is denoted by  $\tilde{\mathbf{F}}' = \mathbf{R}'_\mathbf{c}(\mathbf{F}')$ . Its inverse is given by

$$f'_{ij} = \tilde{f}'_{(i-\mathbf{c}_j)j}. \quad (10.12)$$

Denote by  $\mathbf{R}'_\mathbf{c}^{-1}(\cdot)$  this inverse mapping. Finally, we note that for any vector  $\boldsymbol{\lambda}$  of length  $2^m d$ , we can think of this vector as  $\boldsymbol{\lambda} = \text{vec}(\mathbf{F})$ . Then we let  $\tilde{\boldsymbol{\lambda}} = \mathbf{R}'_\mathbf{c}(\boldsymbol{\lambda})$  where  $\tilde{\boldsymbol{\lambda}} := \text{vec}(\tilde{\mathbf{F}})$  and  $\tilde{\mathbf{F}} := \mathbf{R}'_\mathbf{c}(\mathbf{F})$ .

**Lemma 50** *The relative operation is linear. That is,*

$$\mathbf{R}'_\alpha(\phi_x \mathbf{x} + \phi_y \mathbf{y}) = \phi_x \mathbf{R}'_\alpha(\mathbf{x}) + \phi_y \mathbf{R}'_\alpha(\mathbf{y}).$$

*Further, the relative operator is norm preserving. That is,  $\|\mathbf{R}'_\alpha(\mathbf{x})\| = \|\mathbf{x}\|$ .*

**Proof** Linearity is easy to verify. We note that  $\mathbf{R}'_\alpha(\cdot)$  permutes the input vector based on  $\alpha$ , and therefore the norm is preserved. ■

**Lemma 51** *Let  $y \in \Sigma$ . Then  $\mathbf{L}'(y) = \mathbf{R}'_\beta(\mathbf{L}'(\tau_\beta(y)))$ .*

**Proof** For all  $j = 0, \dots, 2^m - 1$ ,  $P(y|j) = P(\tau_\beta(y)|j - \beta)$ . Therefore

$$\log\left(\frac{1}{P(y|j)}\right) = \log\left(\frac{1}{P(\tau_\beta(y)|j - \beta)}\right).$$

This means that

$$\mathbf{L}'(y)_j = \mathbf{L}'(\tau_\beta(y))_{j-\beta}, \quad (10.13)$$

satisfying the definition of relative vector in (10.10). Therefore  $\mathbf{L}'(\tau_\beta(y)) = \mathbf{R}'_\beta(\mathbf{L}'(y))$ . ■

**Lemma 52** *Let  $\mathbf{c}$  be a valid SPC codeword for length- $d$  check  $\mathbf{h}$ . Then  $\mathbf{F}' \in \mathbb{U}'$  if and only if  $\mathbf{R}'_\mathbf{c}(\mathbf{F}') \in \mathbb{U}'$ .*

**Proof** We can use the same logic in the proof of Lemma 47. Thus we omit the details. ■

**Lemma 53** *Suppose a convex set  $\mathbb{C}$  is such that  $\mathbf{x} \in \mathbb{C}$  if and only if  $\mathbf{R}'_\alpha(\mathbf{x}) \in \mathbb{C}$  for some  $\alpha$ , then  $\Pi_\mathbb{C}(\mathbf{R}'_\alpha(\mathbf{v})) = \mathbf{R}'_\alpha(\Pi_\mathbb{C}(\mathbf{v}))$ .*

**Proof** Our proof is by contradiction. Suppose that the projection of  $\mathbf{R}'_\alpha(\mathbf{v})$  onto  $\mathbb{C}$  is  $\mathbf{u} \neq \mathbf{R}'_\alpha(\Pi_\mathbb{C}(\mathbf{v}))$ . Then  $\mathbf{R}'_\alpha^{-1}(\mathbf{u}) \in \mathbb{C}$  and  $\mathbf{R}'_\alpha^{-1}(\mathbf{u}) \neq \Pi_\mathbb{C}(\mathbf{v})$ . Due to Lemma 50, we have

$$\begin{aligned} \|\mathbf{R}'_\alpha^{-1}(\mathbf{u}) - \mathbf{v}\|_2 &= \|\mathbf{u} - \mathbf{R}'_\alpha(\mathbf{v})\|_2 \\ &< \|\mathbf{R}'_\alpha(\Pi_\mathbb{C}(\mathbf{v})) - \mathbf{R}'_\alpha(\mathbf{v})\|_2 \\ &= \|\Pi_\mathbb{C}(\mathbf{v}) - \mathbf{v}\|_2. \end{aligned}$$

This contradicts the fact that  $\Pi_\mathbb{C}(\mathbf{v})$  is the projection of  $\mathbf{v}$  onto  $\mathbb{C}$ . ■

**Lemma 54** *In Algorithm 2, let  $\mathbf{x}^{(\delta)}$ ,  $\mathbf{z}_j^{(\delta)}$  and  $\boldsymbol{\lambda}_j^{(\delta)}$  be the vectors after the  $\delta$ -th iteration when decoding  $\mathbf{y}$ . Let  $\mathbf{x}^{0,(\delta)}$ ,  $\mathbf{z}_j^{0,(\delta)}$  and  $\boldsymbol{\lambda}_j^{0,(\delta)}$  be the vectors after the  $\delta$ -th iteration when decoding  $\mathbf{y}^0$ . If  $\mathbf{x}^{0,(\delta)} = \mathbf{R}'_\mathbf{c}(\mathbf{x}^{(\delta)})$ ,  $\mathbf{z}_j^{0,(\delta)} = \mathbf{R}'_\mathbf{c}(\mathbf{z}_j^{(\delta)})$  and  $\boldsymbol{\lambda}_j^{0,(\delta)} = \mathbf{R}'_\mathbf{c}(\boldsymbol{\lambda}_j^{(\delta)})$  then  $\mathbf{x}^{0,(\delta+1)} = \mathbf{R}'_\mathbf{c}(\mathbf{x}^{(\delta+1)})$ ,  $\mathbf{z}_j^{0,(\delta+1)} = \mathbf{R}'_\mathbf{c}(\mathbf{z}_j^{(\delta+1)})$  and  $\boldsymbol{\lambda}_j^{0,(\delta+1)} = \mathbf{R}'_\mathbf{c}(\boldsymbol{\lambda}_j^{(\delta+1)})$ .*

**Proof** We drop the iterate  $(\delta)$  for simplicity and denote by  $\mathbf{x}^{\text{new}}$ ,  $\mathbf{z}_j^{\text{new}}$  and  $\boldsymbol{\lambda}_j^{\text{new}}$  the updated vectors at the  $(\delta + 1)$ -th iteration. Let  $\mathbf{x}^0 = \mathbf{R}'_{\mathbf{c}}(\mathbf{x})$ ,  $\mathbf{z}_j^0 = \mathbf{R}'_{\mathbf{c}}(\mathbf{z}_j)$  and  $\boldsymbol{\lambda}_j^0 = \mathbf{R}'_{\mathbf{c}}(\boldsymbol{\lambda}_j)$ . Also let  $\gamma^0$  be the log-likelihood ratio for the received vector  $\mathbf{y}^0$ . From Algorithm 2,

$$\mathbf{x}_i^{0,\text{new}} = \Pi_{\mathbb{S}'_q} \left[ \frac{1}{d_i - \frac{2\alpha}{\mu}} \left( \sum_{j \in \mathcal{N}_v(i)} \left( \mathbf{z}_j^{0,(i)} - \frac{\boldsymbol{\lambda}_j^{0,(i)}}{\mu} \right) - \frac{\gamma_i^0}{\mu} - \frac{2\alpha \mathbf{r}}{\mu} \right) \right]. \quad (10.14)$$

By Lemma 51,  $\gamma_i^0 = \mathbf{R}'_{c_i}(\gamma_i)$ . Then (10.14) can be rewritten as

$$\begin{aligned} \mathbf{x}_i^{0,\text{new}} &= \Pi_{\mathbb{S}'_q} \left[ \frac{1}{d_i - \frac{2\alpha}{\mu}} \left( \sum_{j \in \mathcal{N}_v(i)} \left( \mathbf{R}'_{c_i}(\mathbf{z}_j^{(i)}) - \frac{\mathbf{R}'_{c_i}(\boldsymbol{\lambda}_j^{(i)})}{\mu} \right) - \frac{\mathbf{R}'_{c_i}(\gamma_i)}{\mu} - \frac{2\alpha \mathbf{r}}{\mu} \right) \right] \\ &= \Pi_{\mathbb{S}'_q} [\mathbf{R}'_{c_i}(\mathbf{u})], \end{aligned}$$

where

$$\mathbf{u} = \frac{1}{d_i - \frac{2\alpha}{\mu}} \left( \sum_{j \in \mathcal{N}_v(i)} \left( \mathbf{z}_j^{(i)} - \frac{\boldsymbol{\lambda}_j^{(i)}}{\mu} \right) - \frac{\gamma_i}{\mu} - \frac{2\alpha \mathbf{r}}{\mu} \right).$$

By Lemma 53,

$$\mathbf{x}_i^{0,\text{new}} = \mathbf{R}'_{\mathbf{c}}(\mathbf{x}^{(\delta)}).$$

Let  $\mathbf{v}_j = \mathbf{P}_j \mathbf{x} + \boldsymbol{\lambda}_j / \mu$  and  $\mathbf{v}_j^0 = \mathbf{P}_j \mathbf{x}^0 + \boldsymbol{\lambda}_j^0 / \mu$ , then  $\mathbf{v}_j^{0,(i)} = \mathbf{R}'_{c_i}(\mathbf{v}_j^{(i)})$ . In addition,  $\mathbf{z}_j^{0,\text{new}} = \Pi_{\mathbb{U}'}(\mathbf{v}_j^0)$  and  $\mathbf{z}_j^{\text{new}} = \Pi_{\mathbb{U}'}(\mathbf{v}_j)$ . By Lemma 53,  $\mathbf{z}_j^{0,\text{new}} = \mathbf{R}'_{\mathbf{c}}(\mathbf{z}_j^{\text{new}})$ .

It remains to verify one more equality:

$$\begin{aligned} \boldsymbol{\lambda}_j^{0,\text{new},(i)} &= \boldsymbol{\lambda}_j^{0,(i)} + \mu \left( (\mathbf{P}_j \mathbf{x}^{0,\text{new}})^{(i)} - \mathbf{z}_j^{0,\text{new},(i)} \right) \\ &= \mathbf{R}'_{c_i}(\boldsymbol{\lambda}_j^{(i)}) + \mu \left( (\mathbf{P}_j \mathbf{R}'_{c_i}(\mathbf{x}^{\text{new}}))^{(i)} - \mathbf{R}'_{c_i}(\mathbf{z}_j^{\text{new},(i)}) \right) \\ &= \mathbf{R}'_{c_i}(\boldsymbol{\lambda}_j^{\text{new},(i)}). \end{aligned}$$

■

**Lemma 55** *Let  $\hat{\mathbf{x}} = \mathcal{D}(\mathbf{y})$  be the output of the decoder if  $\mathbf{y}$  is received and  $\hat{\mathbf{x}}^0 = \mathcal{D}(\mathbf{y}^0)$ . Then  $\hat{\mathbf{x}}^0 = \mathbf{R}'_{\mathbf{c}}(\hat{\mathbf{x}})$ .*

**Proof** We note that we initialize Algorithm 2 so that the conditions in Lemma 54 are satisfied. By induction, we always obtain relative vectors at each iteration. It is easy to verify that both decoding processes stop at the same iteration. Therefore  $\hat{\mathbf{x}}^0 = \mathbf{R}'_{\mathbf{c}}(\hat{\mathbf{x}})$ . ■

Due to Lemma 55, if the decoder recovers a codeword  $\mathbf{c}$  from  $\mathbf{y}$ , it means that the decoded vector  $\mathbf{x}$  is the embedding of  $\mathbf{c}$  in the sense of Definition 4. Therefore by (10.9),  $\mathbf{R}'_{\mathbf{c}}(\hat{\mathbf{x}})$  is the embedding of  $\mathbf{0}$  in the sense of Definition 4. This means that the decoder can recover  $\mathbf{0}$  from  $\mathbf{y}^0$ . On the other hand, if the decoder fails to recover codeword  $\mathbf{c}$  from  $\mathbf{y}$ , then  $\mathbf{R}'_{\mathbf{c}}(\hat{\mathbf{x}})$  is not an integral vector. This means that the decoder cannot recover  $\mathbf{0}$  from  $\mathbf{y}^0$ . Combining both arguments, we deduce that the decoder can recover  $\mathbf{c}$  from  $\mathbf{y}$  if and only if it can recover  $\mathbf{0}$  from  $\mathbf{y}^0$ , which completes the proof.



## 11 Linear time $x$ -update algorithm

First note that  $\mathbf{D}^{-1} = \mathbf{D}^T$ , therefore

$$\mathbf{Z}_{j,k}^T \mathbf{Z}_{j,k} = \mathbf{P}_j^T \mathbf{D}_j \mathbf{T}_k^T \mathbf{T}_k \mathbf{D}_j^T \mathbf{P}_j.$$

As a result,

$$\begin{aligned} \mathbf{Z} &= \sum_{j,k} \mathbf{Z}_{j,k}^T \mathbf{Z}_{j,k} \\ &= \sum_{j,k} \mathbf{P}_j^T \mathbf{D}_j \mathbf{T}_k^T \mathbf{T}_k \mathbf{D}_j^T \mathbf{P}_j \\ &= \sum_j \mathbf{P}_j^T \mathbf{D}_j \left( \sum_k \mathbf{T}_k^T \mathbf{T}_k \right) \mathbf{D}_j^T \mathbf{P}_j. \end{aligned}$$

Let  $\mathbf{T} = \sum_k \mathbf{T}_k^T \mathbf{T}_k$ . Then

$$\mathbf{Z} = \sum_j \mathbf{P}_j^T \mathbf{D}_j \mathbf{T} \mathbf{D}_j^T \mathbf{P}_j.$$

We first prove the following Lemma that is useful in studying the structure of  $\mathbf{T}$ .

**Lemma 56** *For an integer  $n \geq 2$ , let  $\tilde{\mathcal{K}}_n = \{\mathcal{K} | \mathcal{K} \subset [n] \text{ and } \mathcal{K} \neq \emptyset\}$ . Let  $\mathbf{u}, \mathbf{v}$  be arbitrary binary vectors of length  $n$  that are not equal to  $\mathbf{0}$ . Then*

1. *The number of sets  $\mathcal{K} \in \tilde{\mathcal{K}}_n$  such that  $\sum_{i \in \mathcal{K}} v_i = 1$  in  $\mathbb{F}_2$  is  $2^{n-1}$ .*
2. *If  $\mathbf{v} \neq \mathbf{u}$ , the number of sets  $\mathcal{K} \in \tilde{\mathcal{K}}_n$  such that  $\sum_{i \in \mathcal{K}} u_i = 1$  and  $\sum_{i \in \mathcal{K}} v_i = 1$  in  $\mathbb{F}_2$  is  $2^{n-2}$ .*

**Proof** We prove both parts of the lemma using inductions.

1. Proof by induction: When  $n = 2$ , the statement holds. Assume that the statement holds for  $n = l$ . Denote by  $\tilde{\mathcal{K}}_{l,0}(\mathbf{v})$  (resp.  $\tilde{\mathcal{K}}_{l,1}(\mathbf{v})$ ) the set of  $\mathcal{K} \in \tilde{\mathcal{K}}_l$  such that  $\sum_{i \in \mathcal{K}} v_i = 0$  (resp.  $\sum_{i \in \mathcal{K}} v_i = 1$ ). This implies

$$|\tilde{\mathcal{K}}_{l,0}(\mathbf{v})| = |\tilde{\mathcal{K}}_{l,1}(\mathbf{v})| = 2^{l-1}.$$

When  $n = l + 1$ , we need to consider binary vectors of length  $l + 1$ . We use  $\mathbf{v}$  to denote the first  $l$  bits of the vector. Let  $x \in \{0, 1\}$  be the  $l + 1$ -th bit of the vector. If  $x = 0$ , then  $\tilde{\mathcal{K}}_{l+1,0}((\mathbf{v}, 0))$  contains the following sets: (a) all sets  $\mathcal{K} \in \tilde{\mathcal{K}}_{l,0}(\mathbf{v})$  and (b) all sets  $\mathcal{K} \cup \{l + 1\}$  where  $\mathcal{K} \in \tilde{\mathcal{K}}_{l,0}(\mathbf{v})$ . Therefore  $|\tilde{\mathcal{K}}_{l+1,0}((\mathbf{v}, 0))| = 2^l$ . Since  $\tilde{\mathcal{K}}_{l+1,1}((\mathbf{v}, 0))$  is the complement of  $\tilde{\mathcal{K}}_{l+1,0}((\mathbf{v}, 0))$ ,  $|\tilde{\mathcal{K}}_{l+1,1}((\mathbf{v}, 0))| = 2^l$ . If  $x = 1$ , then  $\tilde{\mathcal{K}}_{l+1,0}((\mathbf{v}, 1))$  contains the following sets: (a) all sets  $\mathcal{K} \in \tilde{\mathcal{K}}_{l,0}(\mathbf{v})$  and (b) all sets  $\mathcal{K} \cup \{l + 1\}$  where  $\mathcal{K} \in \tilde{\mathcal{K}}_{l,1}(\mathbf{v})$ . Therefore, we get the same result, i.e.

$$|\tilde{\mathcal{K}}_{l+1,0}((\mathbf{v}, 1))| = |\tilde{\mathcal{K}}_{l+1,1}((\mathbf{v}, 1))| = 2^l.$$

Combining the two cases, we conclude that

$$|\tilde{\mathcal{K}}_{l+1,0}((\mathbf{v}, 1))| = 2^l,$$

which completes the proof.

2. Proof by induction. When  $n = 2$ , the statement holds. Assume that the statement holds for  $n = l$ . Denote by  $\tilde{\mathcal{K}}_{l,(0,0)}(\mathbf{u}, \mathbf{v})$  (resp.  $\tilde{\mathcal{K}}_{l,(0,1)}(\mathbf{u}, \mathbf{v})$ ,  $\tilde{\mathcal{K}}_{l,(1,0)}(\mathbf{u}, \mathbf{v})$ ,  $\tilde{\mathcal{K}}_{l,(1,1)}(\mathbf{u}, \mathbf{v})$ ) the set of  $\mathcal{K} \in \tilde{\mathcal{K}}_l$  such that  $\sum_{i \in \mathcal{K}} u_i = 0$  and  $\sum_{i \in \mathcal{K}} v_i = 0$  (the resp. sums equal to  $(0, 1)$ ,  $(1, 0)$  and  $(1, 1)$ ). Note all these sets have  $2^{l-2}$  elements. Similar to the previous proof, we add one more bit to both  $\mathbf{u}$  and  $\mathbf{v}$ . Depending on the combination of these two bits  $((0, 0), (0, 1), (1, 0)$  or  $(1, 1))$ , the four sets  $\tilde{\mathcal{K}}_{l+1,(0,0)}(\mathbf{u}, \mathbf{v})$ ,  $\tilde{\mathcal{K}}_{l+1,(0,1)}(\mathbf{u}, \mathbf{v})$ ,  $\tilde{\mathcal{K}}_{l+1,(1,0)}(\mathbf{u}, \mathbf{v})$  and  $\tilde{\mathcal{K}}_{l+1,(1,1)}(\mathbf{u}, \mathbf{v})$  all double in size. Thus the four new sets all have  $2^{l-1}$  elements. ■

We now describe the structure of  $\mathbf{T}$ .

**Lemma 57**  $\mathbf{T}$  is block diagonal matrix, denoted by

$$\text{diag}(\Phi, \Phi, \dots, \Phi),$$

where

$$\Phi = \begin{pmatrix} 2^{m-1} & 2^{m-2} & 2^{m-2} & \dots & 2^{m-2} \\ 2^{m-2} & 2^{m-1} & 2^{m-2} & \dots & 2^{m-2} \\ \dots & \dots & \dots & \dots & \dots \\ 2^{m-2} & 2^{m-2} & 2^{m-2} & \dots & 2^{m-1} \end{pmatrix}. \quad (11.1)$$

**Proof** The matrices  $\mathbf{T}_k$  are in a one-to-one relation with the set  $\tilde{\mathcal{B}}(\mathcal{K}, 1)$ . Since  $\mathbf{T} = \sum_{k=1}^{2^m-1} \mathbf{T}_k^T \mathbf{T}_k$ ,  $t_{ii} = \sum_{\mathcal{K}} \mathbb{I}(i \in \tilde{\mathcal{B}}(\mathcal{K}, 1))$ , where  $\mathbb{I}$  is the indicator function. In other words,  $t_{ii}$  is the number of sets  $\mathcal{K}$  such that  $i \in \tilde{\mathcal{B}}(\mathcal{K}, 1)$ . Similarly, for all  $i \neq j$ ,  $t_{ij} = \sum_{\mathcal{K}} \mathbb{I}(i, j \in \tilde{\mathcal{B}}(\mathcal{K}, 1))$ . By Lemma 56 we conclude that  $t_{ii} = 2^{m-1}$  and  $t_{ij} = 2^{m-2}$  for all  $1 \leq i, j \leq 2^m - 1$ . ■

Note that  $\mathbf{D}_j$  is also a block diagonal matrix. Therefore

$$\mathbf{D}_j \mathbf{T} \mathbf{D}_j^T = \text{diag}(\mathbf{D}_j(h_1) \Phi \mathbf{D}_j(h_1)^T, \mathbf{D}_j(h_2) \Phi \mathbf{D}_j(h_2)^T, \dots, \mathbf{D}_j(h_d) \Phi \mathbf{D}_j(h_d)^T).$$

**Lemma 58** Let  $\mathbf{D}$  be an  $n \times n$  permutation matrix. Let  $\mathbf{T}$  be an  $n \times n$  matrix whose entries are  $t_{ii} = a$  for all  $i \in [n]$  and some constant  $a$ ,  $t_{ij} = b$  for all  $i \in [n]$ ,  $j \in [n]$  and  $i \neq j$ , and some constant  $b$ . Then,  $\mathbf{D}^T \mathbf{T} \mathbf{D} = \mathbf{T}$ .

**Proof** Without loss of generality, let  $t_{11} = a$  and  $t_{12} = b$ . Let  $\mathbf{X} = \mathbf{D}^T \mathbf{T} \mathbf{D}$ . We need to prove that  $x_{ii} = a$  for all  $i$  and that  $x_{ij} = b$  for all  $i \neq j$ .

Let  $\mathbf{Y} = \mathbf{D}^T \mathbf{T}$ , then  $\mathbf{X} = \mathbf{Y} \mathbf{D}$ . Therefore  $x_{ij} = \sum_{k=1}^n y_{ik} d_{kj}$ . Since  $\mathbf{D}$  is a permutation matrix, there is one 1 in the  $j$ -th column. Without loss of generality assume  $d_{\alpha j} = 1$ . Then  $x_{ij} = y_{i\alpha}$ . Further,  $y_{i\alpha} = \sum_{k=1}^n d_{ki} t_{k\alpha}$ . If  $i \neq j$ , then there exists a  $\beta \neq \alpha$  such that  $\sum_{k=1}^n d_{ki} t_{k\alpha} = t_{\beta\alpha} = t_{12}$ . Thus  $x_{ij} = t_{12} = b$ . If  $i = j$ , note that  $d_{\alpha i} = 1$  and  $d_{ki} = 0$  for all  $k \neq i$ . Therefore  $y_{i\alpha} = \sum_{k=1}^n d_{ki} t_{k\alpha} = t_{\alpha\alpha} = t_{11} = a$ . ■

By Lemma 58 we conclude that  $\mathbf{D}_j \mathbf{T} \mathbf{D}_j^T = \mathbf{T}$  for all  $\mathbf{D}_j$ . As a result,

$$\begin{aligned} \mathbf{Z} &= \sum_j \mathbf{P}_j^T \mathbf{D}_j \mathbf{T} \mathbf{D}_j^T \mathbf{P}_j \\ &= \sum_j \mathbf{P}_j^T \mathbf{T} \mathbf{P}_j \\ &= \text{diag}(d_v \mathbf{T}, d_v \mathbf{T}, \dots, d_v \mathbf{T}), \end{aligned}$$

where  $d_v$  is the variable degree, i.e,  $d_v = |\mathcal{N}_v(i)|$ . Then,

$$\mathbf{Z} + \mathbf{I} = \text{diag}(d_v \mathbf{T} + \mathbf{I}, d_v \mathbf{T} + \mathbf{I}, \dots, d_v \mathbf{T} + \mathbf{I}).$$

Since  $d_v \mathbf{T} + \mathbf{I} = (d_v \mathbf{T} + \mathbf{I})^T$  and its entries only have two values, we can calculate its inverse explicitly. We do this next in Lemma 59.

**Lemma 59** *Let  $\mathbf{T}$  be a  $(2^m - 1)d \times (2^m - 1)d$  block diagonal matrix denoted by  $\text{diag}(\Phi, \Phi, \dots, \Phi)$ , where each  $\Phi$  is the same  $(2^m - 1) \times (2^m - 1)$  matrix in (11.1). Then  $(\mathbf{Z} + \mathbf{I})^{-1} = \text{diag}((d_v \mathbf{T} + \mathbf{I})^{-1}, (d_v \mathbf{T} + \mathbf{I})^{-1}, \dots, (d_v \mathbf{T} + \mathbf{I})^{-1})$  and*

$$(d_v \mathbf{T} + \mathbf{I})^{-1} = \begin{pmatrix} a & b & \dots & b \\ b & a & \dots & b \\ \dots & \dots & \dots & \dots \\ b & b & \dots & a \end{pmatrix},$$

where  $a = \frac{1}{r-s} + b$ ,  $b = \frac{-r}{[r+s(2^m-2)](r-s)}$ ,  $r = 2^m d_v / 2 + 1$  and  $s = 2^m d_v / 4$ .

**Proof** It is easy to verify that the product of the two matrices is the identity matrix. ■

To summarize,  $(\mathbf{Z} + \mathbf{I})^{-1} \mathbf{t}$  can be obtained by calculating each block multiplication due to the fact that  $(\mathbf{Z} + \mathbf{I})^{-1}$  is a block diagonal matrix. For each block we first calculate  $b \|\mathbf{t}_i\|_1$ , where  $\mathbf{t}_i$  is the  $i$ -th block of vector  $\mathbf{t}$ . We then calculate  $(a - b) \mathbf{t}_i + b \|\mathbf{t}_i\|_1$ . This algorithm for  $x$ -update has complexity  $O(q^2 N)$ .

## 12 Results and a conjecture for $\mathbb{F}_{2^2}$

**Lemma 60** *In  $\mathbb{F}_{2^2}$ ,  $\mathbf{F}$  is a valid embedded matrix for the all-ones check if and only if  $\mathbf{F}$  satisfies the first two conditions of Definition 7 and the rows of  $\mathbf{F}$  have either all odd parity **or** all even parity.*

**Proof** Let  $\mathcal{E}^1$  be the set of binary matrices satisfying conditions in this lemma. First, we show  $\mathbf{F} \in \mathcal{E}^1$  implies  $\mathbf{F} \in \mathcal{E}$ , where  $\mathcal{E}$  is defined in Definition 7. Since the field under consideration is  $\mathbb{F}_{2^2}$ ,  $\mathbf{F}$  has three rows. Denote by  $\mathbf{f}_i^R$  the  $i$ -th row of  $\mathbf{F}$ . Then  $\mathbf{g}^1 = \mathbf{f}_1^R + \mathbf{f}_3^R$  and  $\mathbf{g}^2 = \mathbf{f}_2^R + \mathbf{f}_3^R$ . If  $\mathbf{F} \in \mathcal{E}^1$ ,

$$\begin{aligned} \sum_{j=1}^{d_c} g_j^1 &= \sum_{j=1}^{d_c} (f_{1j} + f_{3j}) \\ &= \sum_{j=1}^{d_c} f_{1j} + \sum_{j=1}^{d_c} f_{3j}. \end{aligned}$$

If  $\sum_{j=1}^{d_c} f_{1j} = 1$ , then  $\sum_{j=1}^{d_c} f_{3j} = 1$  by the definition of  $\mathcal{E}^1$ , which means that the overall sum is 0 in  $\mathbb{F}_2$ . If both the sums have even parity, then the overall sum is also 0. It is easy to verify that the same situation holds for  $\mathbf{g}^2$ . Thus  $\mathbf{F} \in \mathcal{E}$ .

Now we show that  $\mathbf{F} \in \mathcal{E}$  implies  $\mathbf{F} \in \mathcal{E}^1$ . Let  $\mathbf{F} \in \mathcal{E}$ . We need to show that if  $\mathbf{f}_1$  has odd (even) parities,  $\mathbf{f}_2$  and  $\mathbf{f}_3$  must also have odd (even) parities. This can be proved by contradiction.

Suppose  $\mathbf{f}_1$  has odd parity and  $\mathbf{f}_2$  (or  $\mathbf{f}_3$ ) has even parity, then  $\mathbf{g}^1 = \mathbf{f}_1 + \mathbf{f}_3$  (or  $\mathbf{g}^2 = \mathbf{f}_2 + \mathbf{f}_3$ ) does not have even parity, contradicting the assumption that  $\mathbf{F} \in \mathcal{E}$ . Similarly, if  $\mathbf{f}_1$  has even parity, the other two rows must both have even parity. ■

In  $\mathbb{F}_{2^2}$ , considering the all-ones check, the characteristics of the polytope in Definition 15 can be simplified to the following:

**Definition 61** Denote by  $\mathbb{U}_4$  the relaxed code polytope for  $\mathbb{F}_{2^2}$  for the all-ones checks. A  $3 \times d_c$  matrix  $\mathbf{F} \in \mathbb{U}_4$  if the following constraints hold:

1.  $f_{ij} \in [0, 1]$ .
2.  $\sum_{i=1}^3 f_{ij} \leq 1$ .
3. Let  $\mathbf{f}_i$  be the  $i$ -th row of  $\mathbf{F}$ . Let  $\mathbf{g}^1 = \mathbf{f}_1 + \mathbf{f}_3$ ,  $\mathbf{g}^2 = \mathbf{f}_2 + \mathbf{f}_3$  and  $\mathbf{g}^3 = \mathbf{f}_1 + \mathbf{f}_2$ . Then  $\mathbf{g}^1 \in \mathbb{P}_{d_c}$ ,  $\mathbf{g}^2 \in \mathbb{P}_{d_c}$  and  $\mathbf{g}^3 \in \mathbb{P}_{d_c}$ .

**Conjecture 62** Let  $\mathcal{E}$  be defined by Lemma 12 for  $\mathbb{F}_{2^2}$  and by the all-ones check of length  $d$ . Let  $\mathbb{U}_4$  be defined by Definition 61. Then  $\mathbb{U}_4 = \text{conv}(\mathcal{E})$ .

We validated this result numerically using the following approach. We randomly generated  $10^6$  vectors of length  $d$  where the entries of each vector are i.i.d. uniformly distributed in  $[0, 4]$ . We then projected these vectors onto both  $\mathbb{U}_4$  and  $\text{conv}(\mathcal{E})$  using CVX (cf. [30]). What we observed is that the projections onto these two polytopes are the same for every vector. Therefore we believe that the conjecture should hold. However, a proof of the conjecture remains open.

### 13 Calculating $E_s/N_0$

In QPSK, the coded symbols are  $0, 1, \xi^1, \xi^2$ . We use vectors  $(1, 0)$ ,  $(0, 1)$ ,  $(-1, 0)$  and  $(0, -1)$  to represent the modulated signals. Let  $\mathbf{x}$  be the modulated vector of length  $2N$ , where  $N$  is the block length. Then the received signal is  $\mathbf{y} = \mathbf{x} + \mathbf{n}$ . Where  $\mathbf{n}$  is i.i.d. Gaussian of variance  $\sigma^2$ .

Let  $N_0$  be the noise spectrum density. The noise is equivalent to have variance of  $\sigma^2 = N_0/2$  per dimension. Let  $R$  be the symbol rate and fix  $E_s = 1/R$ . Let  $\gamma = 10^{\frac{E_s/N_0}{10}}$  be the (non-dB) value for  $E_s/N_0$ , then

$$\sigma^2 = \frac{1}{2\gamma R}.$$

Suppose we use the length-80 code<sup>11</sup> defined in [16] which has rate  $R = 0.6$ . Then  $E_s/N_0 = 4\text{dB}$  translates into  $\sigma = \sqrt{\frac{1}{2 \times 10^{\frac{4}{10}} \times 0.6}} = 0.5760$ .

## References

- [1] M. Breithach, M. Bossert, R. Lucas, and C. Kemper, "Soft-decision decoding of linear block codes as optimization problem," *Eur. Trans. Telecommun.*, vol. 9, no. 3, pp. 289–293, 1998.
- [2] J. Feldman, M. Wainwright, and D. Karger, "Using linear programming to decode binary linear codes," *IEEE Trans. Inf. Theory*, vol. 51, no. 3, pp. 954–972, Mar. 2005.

---

<sup>11</sup>The parity-check matrix of this code is denoted by  $\mathcal{H}^{(2)}$  in [16].

- [3] J. Feldman, T. Malkin, R. A. Servedio, C. Stein, and M. J. Wainwright, “LP decoding corrects a constant fraction of errors,” *IEEE Trans. Inf. Theory*, vol. 53, no. 1, pp. 82–89, Jan. 2007.
- [4] C. Daskalakis, A. Dimakis, R. Karp, and M. Wainwright, “Probabilistic analysis of linear programming decoding,” *IEEE Trans. Inf. Theory*, vol. 54, no. 8, pp. 3565–3578, Aug. 2008.
- [5] S. Arora, C. Daskalakis, and D. Steurer, “Message-passing algorithms and improved LP decoding,” *IEEE Trans. Inf. Theory*, vol. 58, no. 12, pp. 7260–7271, Dec. 2012.
- [6] N. Halabi and G. Even, “LP decoding of regular LDPC codes in memoryless channels,” *IEEE Trans. Inf. Theory*, vol. 57, no. 2, pp. 887–897, Feb. 2011.
- [7] L. Bazzi, B. Ghazi, and R. Urbanke, “Linear programming decoding of spatially coupled codes,” *IEEE Trans. Inf. Theory*, vol. 60, no. 8, pp. 4677–4698, Aug. 2014.
- [8] P. O. Vontobel and R. Koetter, “Towards low complexity linear programming decoding,” in *Proc. Int. Symp. Turbo Codes and Related Topics*, Munich, Germany, Apr. 2006, pp. 1–9.
- [9] M.-H. Taghavi and P. Siegel, “Adaptive methods for linear programming decoding,” *IEEE Trans. Inf. Theory*, vol. 54, no. 12, pp. 5396–5410, Dec. 2008.
- [10] D. Burshtein, “Iterative approximate linear programming decoding of LDPC codes with linear complexity,” *IEEE Trans. Inf. Theory*, vol. 55, no. 11, pp. 4835–4859, Nov. 2009.
- [11] M. Taghavi, A. Shokrollahi, and P. Siegel, “Efficient implementation of linear programming decoding,” *IEEE Trans. Inf. Theory*, vol. 57, no. 9, pp. 5960–5982, Sept. 2011.
- [12] S. Barman, X. Liu, S. C. Draper, and B. Recht, “Decomposition methods for large scale LP decoding,” *IEEE Trans. Inf. Theory*, vol. 59, no. 12, pp. 7870–7886, Dec. 2013.
- [13] X. Zhang and P. H. Siegel, “Efficient iterative LP decoding of LDPC codes with alternating direction method of multipliers,” in *IEEE Int. Symp. Inf. Theory (ISIT)*, Istanbul, Turkey, July 2013, pp. 1501–1505.
- [14] G. Zhang, R. Heusdens, and W. Kleijn, “Large scale LP decoding with low complexity,” *IEEE Commun. Lett.*, vol. 17, no. 11, pp. 2152–2155, Oct. 2013.
- [15] X. Liu and S. C. Draper, “The ADMM penalized decoder for LDPC codes,” *ArXiv preprint 1409.5140*, Sept. 2014.
- [16] M. Flanagan, V. Skachek, E. Byrne, and M. Greferath, “Linear-programming decoding of nonbinary linear codes,” *IEEE Trans. Inf. Theory*, vol. 55, no. 9, pp. 4134–4154, Sept. 2009.
- [17] D. Declercq and M. Fossorier, “Decoding algorithms for nonbinary LDPC codes over  $GF(q)$ ,” *IEEE Trans. Commun.*, vol. 55, no. 4, pp. 633–643, Apr. 2007.
- [18] D. Goldin and D. Burshtein, “Iterative linear programming decoding of nonbinary LDPC codes with linear complexity,” *IEEE Trans. Inf. Theory*, vol. 59, no. 1, pp. 282–300, Jan. 2013.
- [19] M. Punekar, P. Vontobel, and M. Flanagan, “Low-complexity LP decoding of nonbinary linear codes,” *IEEE Trans. Commun.*, vol. 61, no. 8, pp. 3073–3085, Aug. 2013.

- [20] J. Honda and H. Yamamoto, “Fast linear-programming decoding of LDPC codes over  $GF(2^m)$ ,” in *Int. Symp. Inf. Theory and its Apps. (ISITA)*, Honolulu, HI, USA, Oct. 2012, pp. 754–758.
- [21] B. Touri, “Some new results on linear programming decoding,” Master’s thesis, Jacobs University Bremen, Bremen, Germany, 2008.
- [22] X. Zhang and P. Siegel, “Adaptive cut generation algorithm for improved linear programming decoding of binary linear codes,” *IEEE Trans. Inf. Theory*, vol. 58, no. 10, pp. 6581–6594, Oct. 2012.
- [23] J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra, “Efficient projections onto the  $\ell_1$ -ball for learning in high dimensions,” in *Proc. Int. Conf. on Machine Learning (ICML)*, Helsinki, Finland, July 2008, pp. 272–279.
- [24] A. W. Marshall, I. Olkin, and B. C. Arnold, *Inequalities: Theory of majorization and its applications*. Springer, 2009.
- [25] X. Liu and S. C. Draper, “ADMM decoding of non-binary LDPC codes in  $\mathbb{F}_{2^m}$ ,” in *IEEE Int. Symp. Inf. Theory (ISIT)*, Honolulu, HI, USA, June 2014, pp. 2449–2453.
- [26] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, “Distributed optimization and statistical learning via the alternating direction method of multipliers,” *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, July 2011.
- [27] H. Wang and A. Banerjee, “Online alternating direction method,” in *Proc. Int. Conf. on Machine Learning (ICML)*, Edinburgh, Scotland, UK, July 2012.
- [28] D. J. C. MacKay, “Encyclopedia of sparse graph codes,” available online at <http://www.inference.phy.cam.ac.uk/mackay/codes/data.html>.
- [29] R. M. Tanner, D. Sridhara, and T. Fuja, “A class of group-structured LDPC codes,” in *Int. Symp. Comm. Theory and Apps. (ISCTA)*, Ambleside, UK, July 2001.
- [30] M. Grant and S. Boyd, “CVX: Matlab software for disciplined convex programming, version 2.1,” <http://cvxr.com/cvx>, Mar. 2014.